Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jure Bajt

Optimizacija spletnih strani za mobilne naprave in odzivno spletno oblikovanje

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Narvika Bovcon

Ljubljana, 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Napredek spletnih tehnologij je omogočil razvoj kompleksnih spletnih aplikacij, ki vsakodnevno rešujejo izzive milijardam uporabnikov po vsem svetu. Ti uporabniki se na svetovni splet povezujejo z najrazličnejšimi napravami, zato je potrebno poskrbeti, da spletne strani pravilno delujejo na čim širšem spektru naprav. V diplomski nalogi predstavite strategije, ki omogočajo razvoj tako prilagojenih spletnih strani. Opišite tehnike, s katerimi je mogoče doseči uporabnost, dostopnost, trajnost in zmogljivost spletnih strani. Uporabo predstavljenih načinov optimizacije prikažite na praktičnem primeru izdelave spletne strani, prilagojene za mobilne naprave.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Jure Bajt sem avtor diplomskega dela z naslovom:

Optimizacija spletnih strani za mobilne naprave in odzivno spletno oblikovanje (angl. *Website optimization for mobile devices and responsive web design*)

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Narvike Bovcon,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 14. marca 2016 Podpis avtorja:

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
WWW	World Wide Web	svetovni splet
HTML	Hyper-Text Markup Language	hipertekstovni označevalni jezik
CSS	Cascading Style Sheets	prekrivni slogi
HTTP	Hyper-Text Transfer Protocol	protokol za prenos hiperteksta
DPI	Dots Per Inch	število slikovnih pik na palec
ARIA	Accessible Rich Internet Applications	dostopne obogatene spletne aplikacije
UA	User Agent	uporabniški agent
API	Application Programming Interface	vmesnik za programiranje
SVG	Scalable Vector Graphics	skalabilna vektorska grafika
DNS	Domain Name System	sistem domenskih imen
DOM	Document Object Model	model za predstavitev objektov dokumenta

Povzetek

Naslov: Optimizacija spletnih strani za mobilne naprave in odzivno spletno oblikovanje

V letu 2016 si življenja brez svetovnega spleta verjetno sploh ne moremo več predstavljati. Napredek spletnih tehnologij je omogočil razvoj kompleksnih spletnih aplikacij, ki vsakodnevno rešujejo izzive milijardam uporabnikov po vsem svetu. Ti uporabniki se na svetovni splet povezujejo z najrazličnejšimi napravami, zato je potrebno poskrbeti, da spletne strani pravilno delujejo na čim širšem spektru naprav. V diplomski nalogi so opisane strategije in najboljše prakse, ki omogočajo razvoj tako prilagojenih spletnih strani. Podrobneje so predstavljeni načini izdelave odzivnega izgleda spletne strani in načini za zagotavljanje uporabnosti ter dostopnosti spletne strani. Prav tako je v nalogi obravnavan vidik trajnosti spletne strani, ki je v času, ko se tehnologije svetovnega spleta tako hitro razvijajo, zelo pomemben. V zadnjem delu diplomske naloge je pozornost namenjena zmogljivosti spletne strani, ki je nujna za dobro uporabniško izkušnjo njenih uporabnikov.

Ključne besede: optimizacija spletnih strani, odzivno spletno oblikovanje, uporabnost, dostopnost, trajnost, zmogljivost, mobilne naprave.

Abstract

Title: Website optimization for mobile devices and responsive web design

In 2016, we could probably no longer imagine life without the World Wide Web. Advances in web technologies have enabled the development of complex web applications that solve the issues of billions of users on a daily basis. These users connect to the World Wide Web using all types of devices making it necessary to ensure that all websites work properly on a broad as possible spectrum of devices. In our diploma thesis we have described the strategies and best practices that enable the development of these adapted websites. We have presented in more detail the means of creating a responsive website design and how to ensure its usability, as well as the accessibility of the website. We have furthermore dealt with the aspect of a website's sustainability, which, in a time when web technologies are developing so quickly, is very important. In the last part of our diploma thesis we have focused on the performance capabilities of websites, which are vital for providing its users with a good user experience.

Keywords: website optimization, responsive web design, usability, accessibility, sustainability, performance, mobile devices.

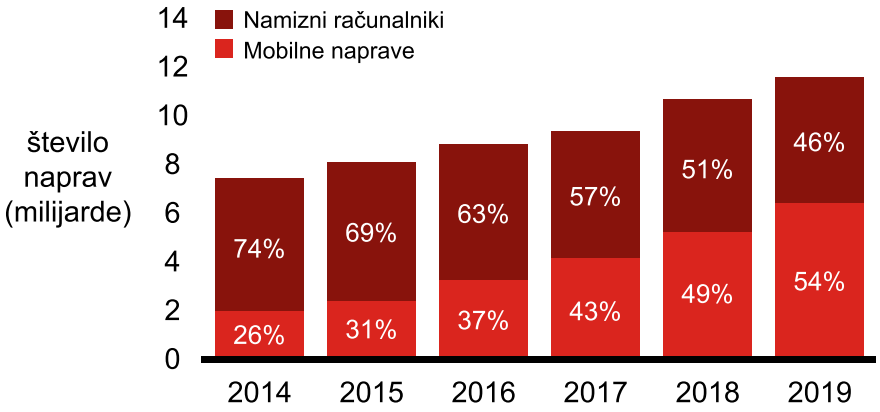
Poglavje 1

Uvod

Začetki svetovnega spleta segajo v leto 1989. Tistega leta je Tim Berners-Lee, britanski znanstvenik v CERN-u, na svojem računalniku postavil prvo spletno stran – predstavitev projekta World Wide Web (v nadaljevanju WWW). Stran je bila namenjena opisu osnovnih funkcionalnosti spleta in objavi navodil, kako dostopati do objavljenih dokumentov druge osebe ter kako postaviti svoj lastni spletni strežnik. Glavna ideja projekta WWW je bila lažja komunikacija in deljenje informacij med znanstveniki, univerzami in drugimi raziskovalnimi ter izobraževalnimi ustanovami po svetu. 30. aprila 1993 je ustanova CERN standard WWW odprla za javnost. [3]

Skozi leta se je tehnologija svetovnega spleta tako razvila, da si v letu 2016 življenja brez svetovnega spleta verjetno sploh ne moremo več predstavljati. Razvoj je potekal od preprostih besedilnih spletnih strani, do katerih so dostopali le redki uporabniki, ki so imeli dostop do računalnika, povezanega v svetovni splet, do kompleksnih spletnih aplikacij, ki vsakodnevno rešujejo najrazličnejše izzive milijardam uporabnikov po vsem svetu.

Sprva so uporabniki za dostop do svetovnega spleta uporabljali le računalnike, leta tehnološkega napredka pa so privedla do tega, da se v svetovni splet lahko povezujemo z najrazličnejšimi mobilnimi napravami (mobilni telefoni, tablice, ure, e-bralniki, potovalni računalniki v avtomobilih ipd.). V letu 2014 je število naprav, ki se povezujejo v svetovni splet,



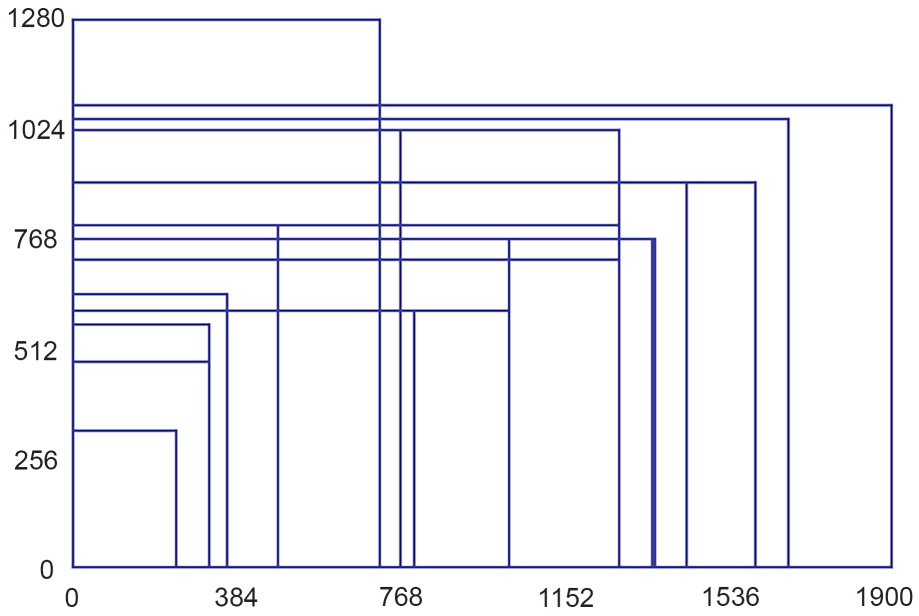
Slika 1.1: Razmerje povezav v svetovni splet preko namiznimih računalnikov in mobilnih naprav v letih od 2014 do 2019.

preseglo število prebivalcev Zemlje, statistične analize pa napovedujejo še večjo rast uporabe mobilnih naprav za dostop do svetovnega spleta. Kot je razvidno iz grafa na sliki 1.1 je za leto 2019 predvideno, da bo število mobilnih naprav, povezanih v svetovni splet, preseglo število računalnikov s povezavo do svetovnega spleta. [4]

Vsaka izmed različnih naprav ima svoje tehnične specifikacije, svoje načine reševanja tehnoloških izzivov, svoje načine prikaza podatkov, uporabe ipd. Naprave delujejo v različnih pogojih, v svetovni splet se povezujejo preko različno zmogljivih in različno hitrih povezav. Že samo pogled na grafični prikaz različnih velikosti zaslonov 20 najbolj široko uporabljenih mobilnih naprav na sliki 1.2 priča o velikih razlikah med napravami, velikost zaslona pa je samo ena od mnogih lastnosti, po katerih se naprave razlikujejo med seboj. [5]

1.1 Odzivno spletno oblikovanje

V današnjem času je eden največjih izzivov spletnih razvijalcev prav razvoj spletnih strani, ki bodo pravilno delovale na čim širšem spektru naprav – od



Slika 1.2: Grafični prikaz različnih velikosti zaslonov 20 najbolj široko uporabljenih mobilnih naprav. Vir: [5]

namiznih računalnikov do najnovejših pametnih ur in ostalih mobilnih naprav. Reševanje tega izziva se je sprva odražalo v oblikovanju takih spletnih strani, katerih izgled (postavitev komponent, velikost le-teh ipd.) se prilagaja velikosti zaslona naprave, ki jo uporabnik uporablja za dostop do strani. V letu 2010 je Ethan Marcotte ta pristop k oblikovanju spletnih strani poimenoval odzivno spletno oblikovanje. [2] Pristop v osnovi temelji na uporabi tekoče postavitve elementov, tekoče velikosti slik in ostalih medijskih objektov ter medijskih poizvedb. Omogoča izdelavo prilagodljivih oblik spletnih strani, tako da le-te izgledajo kar se da dobro na čim več napravah. Vendar so prilagodljive oblike strani le začetek. Da bi uporabnikom spletnih strani zagotovili dobro uporabniško izkušnjo, ne glede na napravo, s katero do strani dostopajo, je potrebno še veliko več – spletno stran je potrebno optimizirati za prikaz in delovanje na vseh teh napravah.

1.2 Optimizacija spletnih strani za širok spekter naprav

Da lahko govorimo o optimizirani spletni strani, mora le ta poleg prilagodljivega izgleda zadostiti še sledečim točkam: [1]

1. Uporabnost: Izgled uporabniškega vmesnika in način interakcije z uporabniškim vmesnikom morata biti prilagojena lastnostim naprave in načinu interakcije z napravo.

2. Dostopnost: Uporabniki vseh naprav, brskalnikov in asistivnih tehnologij lahko dostopajo in razumejo funkcije ter vsebino spletne strani.

3. Trajnost: Sposobnost tehnologij, ki poganjajo spletno stran, da pravilno delujejo danes in da bodo uporabne in dostopne tudi uporabnikom in napravam v prihodnosti.

4. Zmogljivost: Percepcija hitrosti, s katero se vsebina in funkcionalnost spletne strani preneseta k uporabniku in odzivnost/hitrost delovanja uporabniškega vmesnika.

V okviru praktičnega dela diplomske naloge smo zgoraj naštete principe prilagajanja in optimizacije spletnih strani poizkusili čim bolje upoštevati pri načrtovanju in izdelavi prenovljenega čelnega dela aplikacije za spletno prodajo vstopnic mojekarte.si, ki je v lasti podjetja Programski atelje A&Z d.o.o. Primeri uporabe in implementacije principov optimizacije spletnih strani in odzivnega spletnega oblikovanja zato ponekod rešujejo povsem specifične izzive, ki nastopijo pri izdelavi aplikacije za prodajo in posredovanje vstopnic.

Poglavje 2

Uporabljene tehnologije

2.1 HTML5

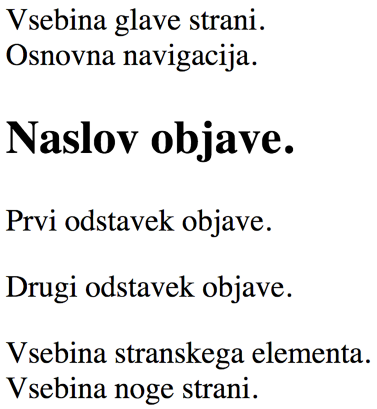
HTML5 je označevalni jezik, ki predstavlja osnovno tehnologijo svetovnega spleta. Uporabljen je za definicijo strukture in vsebine spletnih strani. Spletni brskalnik razčleni html dokument in prikaže uporabniku enostaven in razumljiv izris spletne strani.

Strukturo strani in njeno funkcionalnost definirajo posamezni elementi, ki so v HTML dokumentu opisani z značkami. Primeri značk so: *main*, *header*, *nav*, *section*, *article*, *aside*, *footer*, *figure*, *audio* ipd. Spletni brskalnik zna interpretirati in prikazati vsebino posameznih elementov. Poleg tega značke vsebini elementa dodajo semantični pomen. Tako lahko npr. algoritmi spletnih iskalnikov razumejo, kaj je pomen določene spletne vsebine v sklopu celotne strani, in s tem zelo olajšajo uporabnikovo iskanje informacij v svetovnem spletu.

Primer osnovne vsebine HTML dokumenta, ki ga napiše spletni razvijalec:

```
<!doctype html>
<html>
    <head>
        <meta charset="utf-8">
        <title>HTML5</title>
```

```
        <meta name="description"
            content="Primer osnovnega html dokumenta.">
    </head>
    <body>
        <header class="main-header">
            Vsebina glave strani.
        </header>
        <nav class="main-navigation">
            Osnovna navigacija.
        </nav>
        <main class="main-content">
            <article>
                <header>
                    <h1>Naslov objave.</h1>
                </header>
                <p>
                    Prvi odstavek objave.
                </p>
                <p>
                    Drugi odstavek objave.
                </p>
            </article>
            <aside class="main-sidebar">
                Vsebina stranskega elementa.
            </aside>
        </main>
        <footer class="main-footer">
            Vsebina noge strani.
        </footer>
    </body>
</html>
```



Slika 2.1: Primer izrisa enostavne HTML5 kode v brskalniku.

Spletni brskalnik bi zgornji dokument izrisal na način, prikazan na sliki 2.1. Kot lahko opazimo, je izris zelo enostaven in ne spominja ravno na izris spletnih strani, ki jih vsakodnevno obiskujemo. HTML označevalni jezik brskalniku poda le potrebne informacije o strukturi, semantičnem pomenu in vsebini. Če želimo vsebino narediti bolj uporabno in prijetno za pogled, pa moramo uporabiti druge spletne tehnologije, ki jih bomo opisali v nadaljevanju poglavja.

2.2 Prekrivni slogi (CSS)

Prekrivni slogi (ang. Cascading Style Sheets), v nadaljevanju CSS, opisujejo predstavitev dokumenta, napisanega v označevalnem jeziku. V poglavju 2.1 smo pokazali, kako brskalnik izriše osnovni HTML dokument. Z uporabo jezika CSS brskalniku podamo dodatna pravila, kako naj vsebino dokumenta izriše. CSS pravila npr. določajo postavitve elementov, barve, tipografijo, animacije ipd. in tako omogočajo izdelavo najrazličnejših oblik spletnih strani.

Prekrivni slogi tako ločijo pravila za izris od vsebine spletne strani in s tem zagotavljajo večji semantični pomen vsebine. Prav tako omogočajo

pripravo več različnih oblik in predstavitev iste vsebine.

Jezik CSS je skupek navodil brskalniku za izris vsebine, ki jih s pomočjo CSS selektorjev določimo posameznim HTML elementom. S CSS selektorji je mogoča izbira HTML elementov na podlagi atributov, kot so ime elementa, id, razred, dodatni podatkovni atributi ipd. Izbranemu elementu nato dodamo pravila za njegov izris, tako kot to prikazuje naslednji izsek CSS kode:

```
body {                          /* CSS selektor */
    padding: 30px;              /* CSS pravilo */
    font-family: 'Verdana';     /* CSS pravilo */
}

.main-header {
    padding: 15px;
    background-color: #da251d;
    color: #ffffff;
    font-size: 20px;
}

.main-navigation {
    padding: 15px;
    background-color: #f2f2f2;
}

.main-content {
    overflow: auto;
    border-right: 1px solid #f2f2f2;
    border-left: 1px solid #f2f2f2;
}

article {
    float: left;
```

```
    width: calc(55% - 60px);
    padding: 30px;
}

article h1 {
    margin: 0;
    font-family: 'Times New Roman';
}

.main-sidebar {
    float: left;
    width: calc(45% - 60px);
    padding: 30px;
}

.main-footer {
    padding: 15px;
    background-color: #333333;
    color: #ffffff;
}
```

Zgornja CSS pravila brskalniku podajo navodila, kako naj HTML dokument izriše. Izris HTML dokumenta iz 2.1 poglavja po uporabi teh CSS pravil izgleda tako, kot to prikazuje slika 2.2.

2.3 JavaScript

JavaScript je objektno usmerjeni tolmačeni programski jezik (ang. Interpreted language). Je najbolj uporabljen skriptni jezik za izdelavo spletnih strani, vse bolj pa se uporablja tudi v drugih okoljih (npr. za programiranje robotov, interneta stvari ipd.). [6] V okviru diplomske naloge je bil JavaScript uporabljen kot skriptni jezik, ki se izvaja v brskalniku uporabnika spletne

Slika 2.2: Primer izrisa enostavne HTML5 kode v brskalniku ob uporabi osnovnih prekrivnih slogov.

strani in omogoča izdelavo interaktivnih in dinamičnih spletnih strani, ki se odzivajo na najrazličnejše dogodke – te dogodke lahko sprožijo uporabniki s svojimi akcijami ali pa so dogodki posledica programskega spreminjanja spletne strani.

2.4 Protokol HTTP/1.1

Protokol HTTP (ang. Hyper-Text Transfer Protocol) je protokol aplikacijske plasti sistema ISO/OSI oz. TCP/IP protokolarnega sklada. Protokol HTTP je najbolj razširjen protokol, uporabljen za prenos podatkov preko svetovnega spleta. Deluje po principu "zahteva – odgovor" (ang. request – response) in predstavlja način za komunikacijo med odjemalci in strežniki. V kontekstu prenašanja spletnih strani prek spleta je odjemalec najpogosteje spletni brskalnik, strežnik pa spletni strežnik, na katerem se spletna stran nahaja. Spletni brskalnik po uporabnikovem vnosu naslova spletne strani na

ta naslov pošlje HTTP zahtevo. V kolikor se na navedenem naslovu nahaja spletni strežnik, le-ta na zahtevo odgovori. V odgovoru strežnik brskalniku vrne status (ali je bila zahteva usprešno obdelana ali je prišlo do napake) in morebitno zahtevano vsebino. [7]

Primer HTTP zahtevka prikazuje sledeči odsek:

```
GET /si.html HTTP/1.1
Host: www.mojekarte.si
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache
Accept: text/html,application/xhtml+xml,application/xml;
q=0.9,image/webp,*/*;q=0.8
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/48.0.2564.82 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8,sl;q=0.6
Cookie: cookiepref=Hide
```

Odgovor strežnika na zgornjo zahtevo v primeru uspešno obdelane zahteve je sledeč:

```
HTTP/1.1 200 OK
Date: Sun, 31 Jan 2016 08:20:21 GMT
Server: Apache/2.2.3 (CentOS)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate,
post-check=0, pre-check=0
Pragma: no-cache
Vary: Accept-Encoding
Content-Encoding: gzip
```

```
Content-Length: 7597
Connection: close
Content-Type: text/html; charset=UTF-8

(vsebina zahtevanega html dokumenta)
```

Poglavje 3

Odzivno spletno oblikovanje

S pojavom mobilnih naprav, sposobnih brskanja po svetovnem spletu, je vse več spletnih razvijalcev začelo spletne strani prilagajati za prikaz na mobilnih napravah. Prva uporabljena rešitev je bila izdelava dveh ločenih različic spletne strani – ena različica prilagojena prikazu na namiznih in prenosnih računalnikih z večjim zaslonom ter druga različica za prikaz na mobilnih napravah. Različici spletne strani sta bili najpogosteje dostopni na dveh ločenih naslovih. Osnovna domena (npr. www.mojastran.si) je kazala na različico strani za prikaz na računalniških zaslonih. V kolikor je uporabnik do te domene dostopal z uporabo mobilne naprave, je bila njegova zahteva preusmerjana na drug naslov (npr. mobilna.mojastran.si), kjer je strežnik serviral različico strani, prilagojeno za prikaz na manjših zaslonih. Sprva je bil tak pristop prilagajanja spletnih strani mobilnim napravam zadovoljiv, vendar se je kmalu izkazalo, da je opisana rešitev neekonomična in da zahteva preveč virov ter časa za vzdrževanje. Nove mobilne naprave na trgu so se namreč vedno bolj razlikovale v prikazu spletnih strani, zato začetni dve različici spletne strani (osnovna in mobilna različica) kmalu nista bili dovolj. Razvoj dodatnih različic pa bi preveč otežil in upočasnil razvoj spletne strani, zato so spletni razvijalci razvili novo rešitev, imenovano odzivno spletno oblikovanje.

Pojem odzivno spletno oblikovanje je leta 2010 skoval Ethan Marcotte

in označuje izdelavo oblik spletnih strani, ki se odzivajo na napravo, ki jo uporabnik uporablja za dostop do spletne strani. To je mogoče z uporabo treh osnovnih tehnik: [2]

- tekoča postavitev elementov,
- prilagodljive slike in ostali medijski objekti na strani (videoposnetki, izrisi na *canvas* elementu ipd.) ter
- modul medijskih poizvedb (ang. Media queries) v CSS3.

Glavna prednost odzivnega spletnega oblikovanje je potreba po samo eni različici spletne strani, ne glede na spekter naprav, s katerimi uporabniki do spletne strani dostopajo. Zgoraj naštete tehnike namreč omogočajo, da se oblika spletne strani avtomatsko prilagodi napravi. V nadaljevanju bomo vsako od tehnik podrobneje opisali in natančneje določili njen prispevek k avtomatski odzivnosti spletne strani.

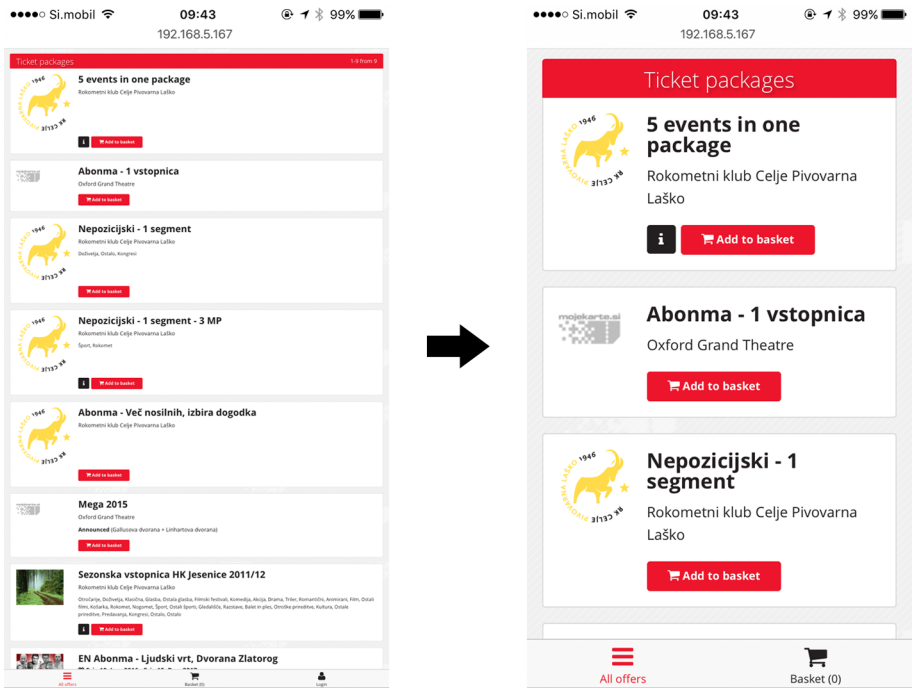
Preden se poglobimo v razlage naštetih tehnik, pa je potrebno izpostaviti še način, kako brskalniku naprave sporočiti, da je obiskana spletna stran odzivna in da naj brskalnik izračun velikosti elementov in prikaz le-teh prepusti strani sami. Prikaz tako izrisane spletne strani prikazuje slika 3.1. To storimo z uporabo t.i. meta oznake v glavi spletne strani: [9]

```
<meta name="viewport" content="width=device-width,
    initial-scale=1.0">
```

Ime oznake *viewport* brskalniku pove, da vsebina meta oznake definira dimenzijo ter povečavo spletne strani. Pomen izrazov, uporabljenih v vsebini meta oznake, je sledeč:

width=device-width Izraz nastavi širino spletne strani na širino zaslona naprave, uporabljene za dostop do strani.

initial-scale=1.0 Izraz nastavi osnovno povečavo prikaza spletne strani ob nalaganju.



Slika 3.1: Izris spletne strani v brskalniku brez oznake (levo) in z vključeno meta oznako *viewport* (desno).

V prihodnosti bo opisana meta oznaka nadomeščena s CSS pravili zapisanimi v spodnjem izseku CSS kode. Zato je dobra praksa, da ta CSS pravila vključimo v spletno stran. [8]

```
@-webkit-viewport{width:device-width}
@-moz-viewport{width:device-width}
@-ms-viewport{width:device-width}
@-o-viewport{width:device-width}
@viewport{width:device-width}
```

3.1 Tekoča postavitev elementov

Velikosti gradnikov spletnih strani, izdelanih brez uporabe tehnik odzivnega spletnega oblikovanja, so največkrat podane z absolutnimi vrednostmi v slikovnih pikah (ang. pixels). Za primer vzemimo element, ki mu s pravili CSS določimo fiksno širino 700 px:

```
.element {
    width: 700px;
}
```

Ta element bo, ne glede na širino brskalniškega okna, vedno enako širok – 700 px. V kolikor je širina brskalniškega okna večja od 700 px, to ne predstavlja težave, saj bo element v celoti viden. Težava se pojavi pri brskalniških oknih s širino, manjšo od 700 px. V tem primeru bo element na določeni širini "odrezan".

Zgoraj opisano težavo lahko rešimo z uporabo relativnih enot za določanje velikosti, kot so npr. odstotki. Definicija velikosti elementa iz prejšnjega odstavka bi z uporabo relativnih enot izgledala takole:

```
.element {
    width: 100%;
    max-width: 700px;
}
```

Širino elementa smo nastavili na 100 % širine predhodnega elementa (v našem primeru je širina predhodnega elementa enaka širini brskalniškega okna). Pravilo CSS *max-width* dodatno določa, da velikost elementa ne sme presegati 700 px, ne glede na širino predhodnega elementa. S tem dosežemo, da bo širina elementa v brskalnikških oknih s širino, manjšo od 700 px, enaka širini okna, v širših brskalniških oknih pa bo element širok natanko 700 px. Element se torej avtomatsko odziva na velikost brskalniškega okna.

Poleg odstotkov poznamo še nekatere druge relativne enote za določanje velikosti. Predvsem pri določanju velikosti pisave je zelo razširjena uporaba enote em. Glavna lastnost te enote je njena odvisnost od velikosti pisave predhodnega elementa. Ta lastnost omogoča lažji razvoj odzivnih spletnih strani, saj se ob prilagoditvi velikosti pisave na nekem elementu posredno prilagodijo tudi izračunane velikosti na vseh elementih znotraj tega elementa po definiranem razmerju.

Primer uporabe enote em:

```
<html>
    <head>
        <style>
            .stars {
                font-size: 1.2em;
            }
            .otrok {
                font-size: 1.2em;
            }
        </style>
    </head>
    <body>
        <p class="stars">
            Besedilo velikosti 19px.
            <span class="otrok">Besedilo velikosti 23px.</span>
        <p>
```

```
    </body>
</html>
```

Velikost pisave elementa *.stars* se izračuna glede na velikost pisave predhodnega elementa, torej elementa *body* in je enaka 19 px. Ker element *body* nima določene velikosti pisave, je velikost pisave elementa *body* enaka velikosti pisave elementa *html*. V večini brskalnikov je privzeta velikost pisave za element *html* 16 px. Izračun velikosti pisave elementa *.stars* je prikazan z izrazom 3.1.

$$round(16px * 1.2) = 19px \tag{3.1}$$

Podobno se tudi velikost pisave elementa *.otrok* izračuna na podlagi velikosti pisave predhodnega elementa, torej elementa *.stars* in je enaka 23 px. Izračun velikosti pisave elementa *.otrok* prikazuje izraz 3.2.

$$round(19px * 1.2) = 23px \tag{3.2}$$

3.2 Prilagodljive slike

V prejšnjem poglavju opisana tehnika tekoče postavitve elementov dobro deluje, ko imamo opravka z besedilnimi spletnimi stranmi. Dandanes pa težko najdemo spletno stran, ki bi vsebovala zgolj besedilo. Velika večina spletnih strani vsebuje poleg besedila še slike, videoposnetke in druge medijske objekte. Težava se pojavi, ko želimo te slike oz. medijske objekte s fiksno širino vstaviti v tekočo postavitev. Vstavljena slika je lahko večja od predhodnega elementa in na ta način podre obliko strani. Rešitev za slike in ostale medijske objekte, ki po velikosti presegajo velikost predhodnega elementa, je, da za vse te objekte dodamo spodnje CSS pravilo, ki jim določa največjo dovoljeno širino. Le-ta je enaka širini predhodnega elementa.

```
img, embed, object, video {
    max-width: 100%;
```

```
}
```

Z zgornjim pravilom smo zagotovili, da slike in ostali medijski objekti niso večji kot oblika spletne strani predvideva in tako oblike strani ne podrejo. Brskalnik bo širino slike nastavil na širino predhodnega elementa, višino pa izračunal tako, da bo razmerje med širino in višino enako razmerju neprilagojene slike. V primeru, da je širina slike manjša od širine predhodnega elementa, bo vstavljena slika izvorne velikosti (slike brskalnik ne bo popačil z raztezanjem).

Pri uporabi prilagodljivih slik pa se pojavi vprašanje: Zakaj s strežnika prenašati večjo sliko (kar posledično pomeni tudi večjo količino bajtov) in jo nato v brskalniku pomanjševati? Ali ne bi bilo bolje s strežnika prenesti sliko manjše velikosti in s tem prihraniti nekaj prenosa podatkov preko omrežja? Izkaže se, da nalaganje najbolj ustrezne velikosti slik izdatno pripomore k optimizaciji spletne strani za mobilne naprave. Podrobneje je navedena tehnika opisana v poglavju 7.6.3.

3.3 Medijske poizvedbe

Tekoča postavitev elementov razvijalcem spletnih strani omogoča enostaven način zagotavljanja odzivnosti spletnih strani. Ta odzivnost pa ni vedno taka, kot bi si jo želeli. Za primer vzemimo obliko postavitve strani, prikazano levo na sliki 3.2.

Uporaba tekoče postavitve elementov bi na napravah z manjšim zaslonom obliko strani avtomatsko prilagodila, tako da bi oblika izgledala, kot prikazuje desna stran slike 3.2. Vidimo lahko, da gradniki postanejo precej ožji, kar lahko predstavlja težavo za prikaz vsebine znotraj teh gradnikov. Z dodatnim manjšanjem zaslona naprave pride ta težava še bolj do izraza.

Rešitev za opisano težavo je prestrukturiranje strani. In ravno to nam omogočajo medijske poizvedbe. Medijske poizvedbe so skupek testov za ugotavljanje lastnosti naprave. Sintaksa medijskih poizvedb je sledeča: [10]

```
@media not|only media_type and (media_feature) {
```

```
    /* CSS pravila, ki se dodajo elementom, ce naprava
       prestane test medijske poizvedbe. */
}
```

Razlaga:

@media Ključna beseda, ki označuje, da je trenutni blok CSS kode del medijske poizvedbe.

not ali only Opcijski parameter, uporaben za npr. negacijo medijske poizvedbe.

media_type Tip medija, možne vrednosti:

- all – vsi mediji,
- screen – prikaz na zaslonu računalnika, tablice, mobilnega telefona ipd.,
- print – oblika za tisk,
- speech – oblika za bralnik zaslona (ang. screenreader).

media_feature Definicija testa za preverjanje določene lastnosti naprave. Najpogosteje uporabljene teste prikazuje tabela 3.1.

Z uporabo medijskih poizvedb lahko strukturo spletne strani iz prejšnjega primera prilagodimo tako, da bo vsebina pravilno prikazana tudi na manjših zaslonih. Spremembo strukture dosežemo s spremembo razmerij med posameznimi gradniki strani. Tako npr. širino gradnikov, predstavljenih z modro barvo, na zaslonih z največjo širino 768 px, spremenimo iz 33 % na 100 %. S tem preprečimo popačenje vsebine zaradi premajhne širine gradnikov. CSS koda za to spremembo je sledeča:

```
.modri {
    width: 100%;
}
```

```
@media all and (min-width: 769px) {
    .modri {
        width: 33%;
    }
}
```

Podobno lahko širine z medijskimi poizvedbami prilagodimo tudi ostalim gradnikom. Popolnoma prilagojena postavitev gradnikov bi na zaslonih različne velikostih izgledala tako, kot prikazuje slika 3.3. Seveda pa to ni edina možna postavitev elementov – kako bomo postavitev gradnikov prestrukturirali, je v največji meri odvisno od vsebine teh gradnikov.

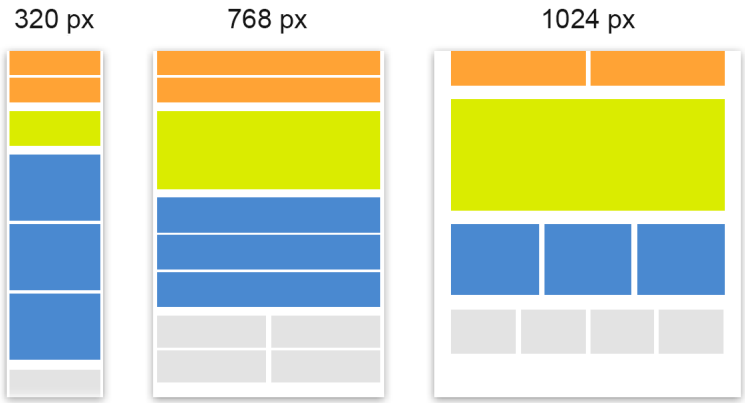
Medijske poizvedbe pa niso uporabne samo pri zaznavanju širine zaslona naprave in pri prestrukturiranju postavitve gradnikov na podlagi te širine. Z uporabo nekaterih testov, prikazanih v tabeli 3.1, lahko naredimo spletne strani še bolj odzivne in prilagojene za velik spekter naprav. Ustrezna medijska poizvedba npr. zazna ločljivost naprave in v primeru, da ima naprava vgrajen visokoločljivostni zaslon, v obliki strani uporabljene slike zamenja z visokoločljivimi različicami, tako da le-te izgledajo ostro na vseh vrstah naprav.

V poglavju 3 smo opisali osnovne tehnike, uporabljene pri odzivnem oblikovanju spletnih strani. Z uporabo omenjenih tehnik zagotovimo, da se oblika strani odziva oz. prilagaja napravi, s katero uporabnik dostopa do spletne strani. Prilagodljive oblike strani pa so le začetek optimizacije spletnih strani za mobilne naprave. Zaradi različnih lastnosti naprav in načinov delovanja ter zaradi omejitev, s katerimi se soočajo naprave in tehnologije svetovnega spleta, so potrebne še številne druge optimizacije, ki jih bomo podrobneje opisali v naslednjih poglavjih.

Ime medijskega testa	Razlaga
height	višina brskalniškega okna (ang. viewport height)
min-height	najmanjša višina brskalniškega okna
max-height	največja višina brskalniškega okna
width	širina brskalniškega okna (ang. viewport width)
min-width	najmanjša širina brskalniškega okna
max-width	največja širina brskalniškega okna
resolution	ločljivost zaslona naprave (dpi)
min-resolution	najmanjša ločljivost zaslona naprave
max-resolution	največja ločljivost zaslona naprave

Tabela 3.1: Najpogosteje uporabljeni medijski testi. Vir: [10]

Slika 3.2: Izris spletne strani s tekočo postavitvijo elementov na širših (levo) in ožjih zaslonih (desno). Vir: [11]



Slika 3.3: Prikaz prilagajanja postavitve gradnikov ob uporabi medijskih poizvedb. Vir: [11]

Poglavje 4

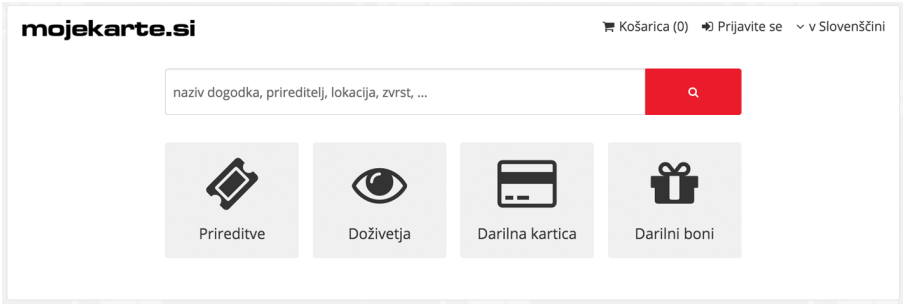
Uporabnost

Uporabnost se v kontekstu odzivnih spletnih strani kaže predvsem v načinu predstavitve vsebine in prilagoditvi funkcionalnosti spletne strani. Vsebino želimo predstaviti na tak način, da bo pravilno predstavljena (pravilno strukturirana, berljiva ipd.) na čim širšem spektru naprav. Prav tako želimo zagotoviti čim boljšo podporo funkcionalnosti spletne strani na velikem številu različnih naprav. V poglavju 4 bomo opisali, kako z uporabo nekaterih tehnik in splošno uveljavljenih najboljših praks izboljšamo uporabnost spletne strani.

4.1 Točke preloma

Kot smo pokazali v poglavju 3.3, lahko s pomočjo medijskih poizvedb prilagajamo strukturo spletne strani tako, da je le-ta najbolj primerna za napravo, s katero je uporabnik obiskal spletno stran. Spremembe postavitve gradnikov spletne strani najpogosteje določamo na podlagi medijskih poizvedb o širini brskalniškega okna (testa *min-width* in *max-width*). Vrednosti, ki jih v testu preverjamo, se imenujejo točke preloma.

Točke preloma so širine brskalniškega okna, pri katerih spremenimo postavitev oz. obliko posameznih gradnikov, tako da zagotovimo pravilni prikaz vsebine. Najpogosteje so točke preloma predstavljene z enoto slikovnih pik

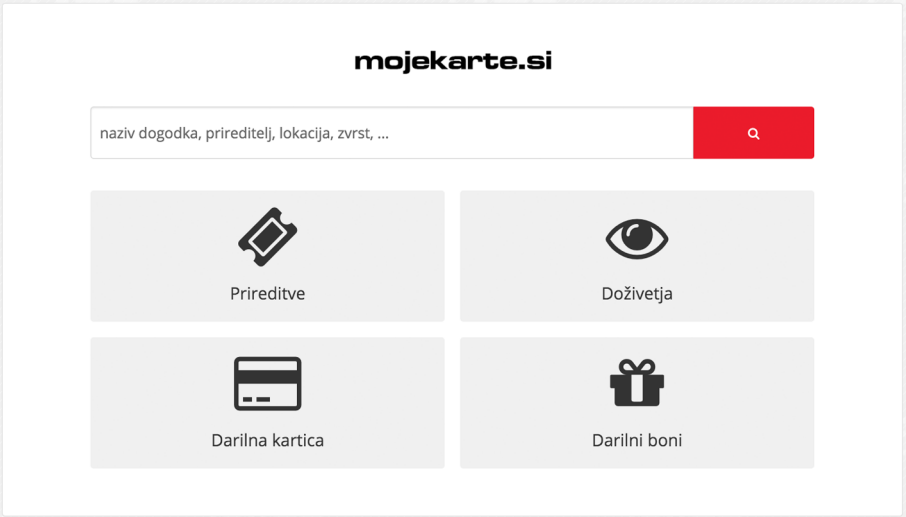


Slika 4.1: Prikaz navigacijskih gumbov na velikih zaslonih.

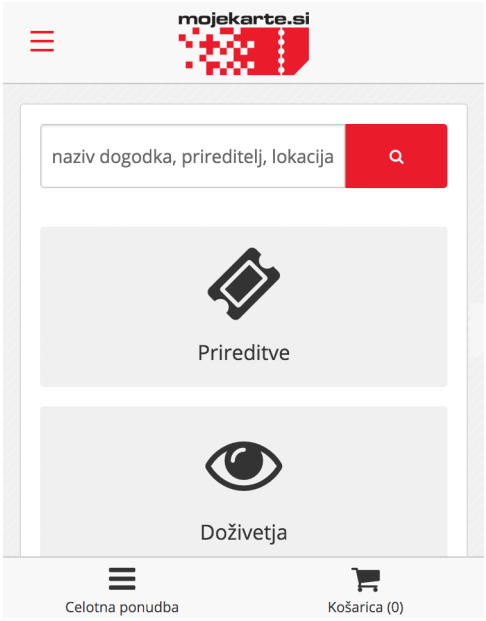
oz. enoto em.

Točke preloma se določajo na dva načina. Prvi način je določanje točk preloma glede na širine zaslonov naprav, za katere želimo prilagoditi prikaz spletne strani. Ta način ni najboljši, saj se število naprav hitro povečuje, s tem pa se povečuje tudi število različnih točk preloma, ki jih je potrebno preverjati z medijskimi poizvedbami. Veliko primernejši za uporabo je način določanja točk preloma na osnovi vsebine spletne strani. Pri tem lahko kot osnovo za pravilen prikaz besedilne vsebine uporabimo pravilo, ki določa, da je optimalno število znakov v vrstici med 45 in 75 znaki. [12] Za ostale gradnike, katerih vsebina primarno ni besedilo, pa lahko pri razvoju s pomočjo spreminjanja širine brskalniškega okna poiščemo širine okna, pri katerih gradnik izgleda nepravilno/popačeno. Te širine brskalniškega okna uporabimo kot točke preloma.

Uporaba opisanega pristopa določanja točk preloma na podlagi vsebine je prikazana na slikah 4.1, 4.2 in 4.3. Postavitev navigacijskih gumbov se prilagaja širini brskalniškega okna. Točke preloma so določene na podlagi vsebine navigacijskih gumbov – na širini, pri kateri bi gumb izgledal popačen, določimo točko preloma in spremenimo strukturo strani. S tem zagotovimo pravilen prikaz navigacijskih gumbov, ne glede na širino brskalniškega okna.



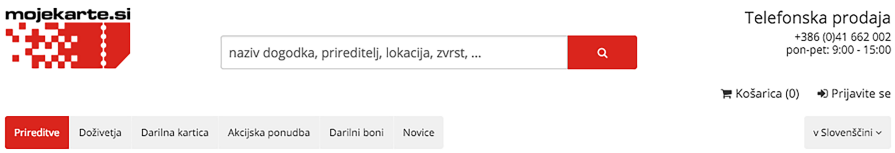
Slika 4.2: Prikaz navigacijskih gumbov na srednje-velikih zaslonih.



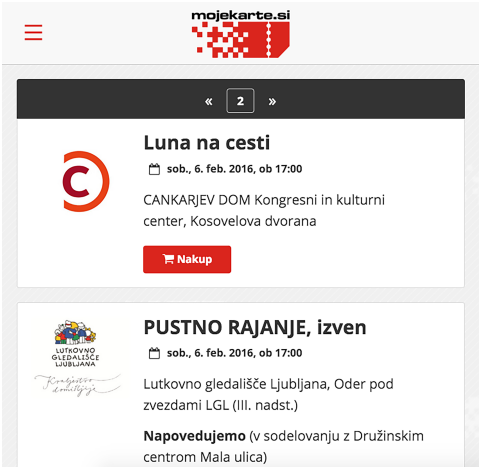
Slika 4.3: Prikaz navigacijskih gumbov na manjših zaslonih.

4.2 Prikazovanje vsebine na zahtevo

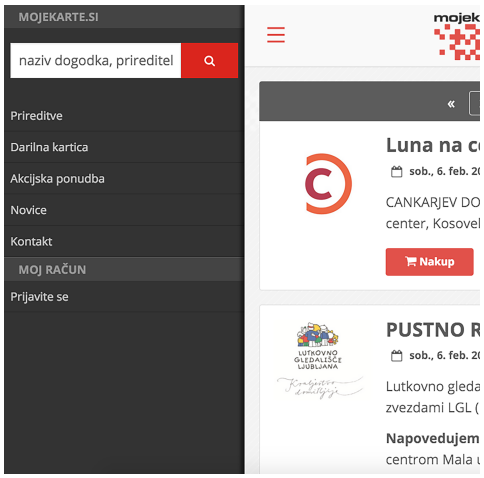
Spremembe postavitve gradnikov spletne strani na podlagi točk preloma predstavljajo učinkovit način zagotavljanja pravilnega prikaza vsebine na širokem spektru naprav. Težava nastane pri veliki količini prikazane vsebine. Kljub temu da je le-ta pravilno prikazana, pomeni manjšo preglednost in uporabnost spletne strani. Logična rešitev te težave je skrivanje "nepotrebne" vsebine. Če pa spletno stran pravilno načrtujemo, lahko kmalu ugotovimo, da "nepotrebne" vsebine ni veliko – če je vsebina dovolj pomembna, da jo prikažemo uporabniku računalnika, je namreč v večini primerov enako pomembna tudi za uporabnika neke mobilne naprave. Zato moramo biti pri skrivanju vsebine pozorni, da je do skrite vsebine vseeno mogoče dostopati. Pri tem nam pomaga tehnika prikazovanja vsebine na zahtevo. Primer uporabe tehnike prikazovanja vsebine na zahtevo je navigacija na spletni strani mojekarte.si. Uporabnikom večjih zaslonov je v glavi spletne strani prikazana celotna navigacija, kot to prikazuje slika 4.4. Le-ta se na manjših zaslonih skrije (slika 4.5), do nje pa je možno dostopati s pritiskom na gumb ob logotipu levo. Ob pritisku na gumb se navigacija pokaže, tako kot to prikazuje slika 4.6. Spletna stran tako ostaja uporabna pri vseh velikostih zaslonov, saj v osnovi ni prenasičena z vsebino, kljub temu pa ima uporabnik možnost dostopati do celotne vsebine.



Slika 4.4: Navigacija in iskalnik v glavi spletne strani prikazana uporabnikom večjih zaslonov.



Slika 4.5: Navigacija in iskalnik v glavi spletne strani sta za uporabnike manjših zaslonov privzeto skrita.



Slika 4.6: Na zahtevo prikazana navigacija in iskalnik.

4.3 Odzivne tabele

Predstavitev podatkov v obliki tabel na manjših zaslonih predstavlja velik izziv. Zato da zagotovimo uporabnost podatkov, predstavljenih v tabeli, je pomembno, da uporabnik v vsakem trenutku poleg podatka v nekem stolpcu vidi tudi glavo stolpca, ki pojasnjuje, kaj stolpec prikazuje. Na manjših zaslonih je temu pogoju težko zadostiti. Višine tabel z več vrsticami so namreč lahko večje od višine zaslona, zato glava take tabele ter vrstica s podatki pogosto nista hkrati prikazani na zaslonu. Prav tako se izrazito zmanjša preglednost tabel, ki vsebujejo večje število stolpcev. Stolpci postanejo preozki za pravilni prikaz vsebine. Poleg tega brskalnik širino stolpcev zmanjša samo do neke vrednosti, po tej vrednosti pa postane tabela neodzivna. To na manjših zaslonih povzroči, da je širina tabele večja od širine zaslona, kar pomeni, da bo del tabele "odrezan", kot prikazuje slika 4.7.

Kot odgovor na opisane težave s prikazom tabel na manjših zaslonih smo razvili rešitev, ki z uporabo HTML in CSS kode tabele prestrukturira tako, da ostanejo pregledne in uporabne tudi na manjših zaslonih. Rešitev za svoje delovanje potrebuje dodatni podatkovni atribut *data-label*, ki za vsako polje v tabeli vsebuje vrednost glave stolpca, v katerem je polje prikazano.

```
<table>
    <thead>
        <tr>
            <th>Številka naročila</th>
            <th>Datum</th>
            <th>Znesek</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td data-label="Številka naročila">67890</td>
            <td data-label="Datum">15.12.2015</td>
```

```
            <td data-label="Znesek">24,00 &euro;</td>
        </tr>
        <tr>
            <td data-label="Številka naročila">12345</td>
            <td data-label="Datum">22.5.2015</td>
            <td data-label="Znesek">23,00 &euro;</td>
        </tr>
    </tbody>
</table>
```

Tabela, zgrajena s pomočje zgornje HTML kode, na večjih zaslonih izgleda tako kot prikazuje slika 4.8. Na manjših zaslonih pa s pomočjo CSS kode tabelo prestrukturiramo tako, kot prikazuje slika 4.9.

Kot je razvidno iz spodnje CSS kode, na zaslonih, ožjih od 401 px, skrijemo glavo tabele. Stolpce v posamezni vrstici razdelimo na več vrstic (vsak stolpec postane nova vrstica), vsebino podatkovnega atributa "data-label"pa z uporabo CSS funkcije *attr()* vstavimo na začetek novonastale vrstice, tako da predstavlja legendo (in nadomešča glavo stolpca), ki opredeljuje, kaj podatek v vrstici pomeni. Iz vključene CSS kode smo zaradi boljše preglednosti izpustili vsa CSS pravila, ki ne vplivajo na postavitev strukture tabele.

```
table {
    width: 100%;
    margin: 0;
    padding: 0;
    border-collapse: collapse;
    border-spacing: 0;
}

@media screen and (max-width: 400px) {
    table thead {
        display: none;
```

```
    }

    table tr {
        display: block;
    }

    table tr:after {
        content: "";
        display: table;
        clear: both;
    }

    table tr:before {
        display: block;
    }

    table td {
        box-sizing: border-box;
        display: block;
        float: left;
        clear: left;
        width: 100%;
    }

    table td:before {
        content: attr(data-label);
        float: left;
    }
}
```

ŠTEVILKA NAROČILA	DATUM	ZNE
67890	15.12.2015	24,
12345	22.5.2015	23,

Slika 4.7: Neodzivne tabele s fiksno širino so na manjših zaslonih "odrezane".

ŠTEVILKA NAROČILA	DATUM	ZNESEK
67890	15.12.2015	24,00 €
12345	22.5.2015	23,00 €

Slika 4.8: Odzivna tabela na večjih zaslonih.

ŠTEVILKA NAROČILA	67890
DATUM	15.12.2015
ZNESEK	24,00 €

ŠTEVILKA NAROČILA	12345
DATUM	22.5.2015
ZNESEK	23,00 €

Slika 4.9: Odzivna tabela prilagojena za prikaz na manjših zaslonih.

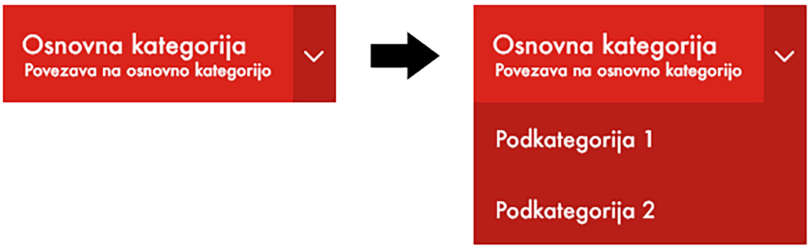
4.4 Rokovanje z različnimi načini vnosa

Z uporabo tehnik, ki smo jih do sedaj opisali v poglavju 4, dosežemo, da se izgled spletne strani in predstavitev njene vsebine prilagajata napravi, s katero uporabnik dostopa do spletne strani. Samo pravilni prikaz vsebine pa uporabniku ne koristi najbolje, če strani ne more uporabljati. Ena izmed ovir, ki uporabniku onemogočajo uporabo spletne strani, je nepravilno rokovanje z različnimi načini vnosa. Uporabniki namiznih in prenosnih računalnikov za interakcijo s spletno stranjo povečini uporabljajo miško in tipkovnico. Uporabniki mobilnih naprav pa praviloma tipkovnice in miške ne uporabljajo, temveč s stranjo upravljajo preko zaslona, občutljivega na dotik. Ravno zato je potrebno poskrbeti, da je vsa interakcija s spletno stranjo, ki temelji na uporabi miškinega kazalca, mogoča tudi ob uporabi zaslona, občutljivega na dotik, ki takega kazalca ne podpira.

Brskalniki naprav, z zasloni občutljivimi na dotik, v večini primerov poskrbijo, da je pritisk na zaslonu na nek element obravnavan tako kot klik z miško na isti element, zato za prepoznavanje teh dogodkov ne potrebujemo dodatne rešitve. Drugače pa je ob prepoznavanju dogodkov, ki so specifični za miškin kazalec. To je predvsem dogodek *hover*, ki se zgodi, ko uporabnik kazalec miške prestavi nad nek element (brez klika). Enakovrednega dogodka na zaslonih na dotik ne poznamo. Potrebno je torej zagotoviti način, s katerim lahko tudi uporabniki naprav z zasloni, občutljivimi na dotik, sprožijo enako akcijo, kot jo sproži dogodek *hover* na napravah, pri katerih uporabniki za interakcijo s spletno stranjo uporabljajo miško.

Primer take prilagoditve je viden na sliki 4.10. Slika 4.10 prikazuje gumb, katerega naloga je prikaz skritega menija, ko uporabnik miškin kazalec prestavi nad ta gumb, ob kliku z miško na gumb pa je uporabnik preusmerjen na stran osnovne kategorije. Uporabniki zaslonov, občutljivih na dotik, lahko do menija podkategorij dostopajo s pritiskom na puščico, ob pritisku na osnovni gumb pa so tako kot uporabniki miške preusmerjeni na stran osnovne kategorije.

Podobno veljajo omejitve pri prepoznavanju vhodnih dogodkov tudi v

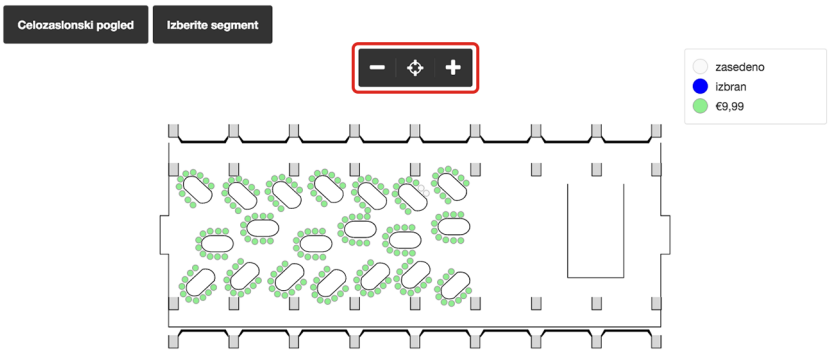


Slika 4.10: Navigacijski gumb s seznamom povezav, ki se prikaže ob ustrezni uporabnikovi akciji.

obratni smeri. Zasloni na dotik omogočajo akcije, ki jih z miško ni mogoče enostavno sprožiti. Primer take akcije je zumiranje z uporabo geste, pri kateri dva prsta, pritisnjena ob zaslon, zbližujemo oz. oddaljujemo. Kot smo že omenili, je potrebno uporabnikom naprav, ki te geste ne podpirajo, omogočiti alternativno akcijo z enakim rezultatom.

Na spletni prodajalni vstopnic mojekarte.si smo se z omenjenim primerom srečali ob izdelavi komponente za izbiro sedežev v dvorani. Uporabniki zaslonov, občutljivih na dotik, lahko za lažjo izbiro sedeža dvorano povečajo oz. pomanjšajo z uporabo geste. Tisti pa, ki za interakcijo z dvorano uporabljajo računalniško miško, lahko enako akcijo sprožijo s klikom na ustrezni kontrolni gumb. Rešitev je prikazana na sliki 4.11, na kateri so kontrolni gumbi označeni z rdečim okvirjem.

Prilagajanje spletne strani napravam z zasloni, občutljivimi na dotik, zahteva poleg navedenega še eno pomembno prilagoditev – elementi, namenjeni interakciji s pomočjo dotika, morajo biti dovolj veliki. Rezultati raziskave, izvedene v "MIT's touch lab", so pokazali, da je priporočljiva velikost teh elementov med 45 in 57 slikovnimi pikami. Za gumbe, primarno namenjene pritiskom s palcem, je priporočena velikost še nekoliko večja, in sicer 72 slikovnih pik. [13] Uporabniku lahko še dodatno olajšamo interakcijo z elementi spletne strani s tem, da okoli elementov pustimo nekaj praznega prostora. S tem zmanjšamo možnost interakcije z napačnim elementom ob nekoliko "zgrešenem" uporabnikovem pritisku.



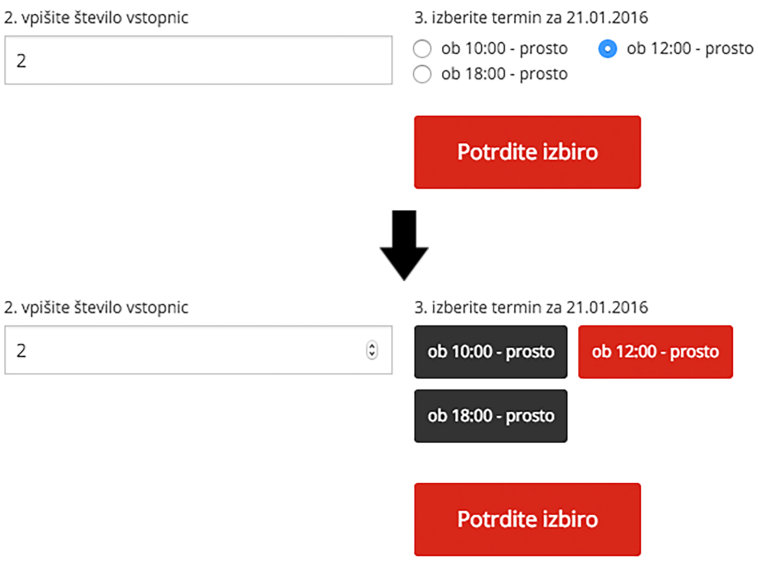
Slika 4.11: Kontrolni gumbi namenjeni nastavljanju povečave izrisa dvorane.

Poglavje 5

Dostopnost

V poglavju 4 smo opisali, kako spletno stran prilagoditi za uporabnost. To dosežemo s prilagajanjem vsebine različnim velikostim zaslona in ustreznim rokovanjem z različnimi načini vnosa. Da pa bi bile posamezne komponente resnično uporabne, je potrebno zagotoviti, da so dostopne tudi uporabnikom brskalnikov, ki ne podpirajo naprednejših načinov predstavitve spletnih strani, ter uporabnikom asistivnih tehnologij.

Dostopnost komponent je najlažje zagotoviti, če so le-te sestavljene z uporabo semantično najustreznejših HTML gradnikov. HTML jezik je načrtovan tako, da je obratno združljiv s svojimi starejšimi verzijami, kar pomeni, da ga bodo povečini pravilno razumele in predstavile tudi naprave brez podpore za naprednejše funkcionalnosti. Pravilno strukturirani HTML dokumenti so zato po svoji naravi zelo dostopni, njihova dostopnost pa se lahko precej zmanjša z nepazljivo uporabo stilnih predlog in z nepazljivim dodajanjem funkcionalnosti s pomočjo JavaScript jezika. Omenjeni težavi se je najlažje izogniti, če se pri razvoju komponente upoštevajo načela postopnega izboljševanja. Ta načela zagovarjajo, da je osnova komponente funkcionalna in semantično pravilna HTML koda. Izgled in funkcionalnost komponente sta nato lahko izboljšana z uporabo stilnih predlog in JavaScripta, vendar na način, ki ne preprečuje uporabe komponente tudi v brskalnikih, ki izboljšane predstavitve ne podpirajo.



Slika 5.1: Prikaz komponente za izbiro termina in števila vstopnic, načrtovane z uporabo tehnike za postopno izboljševanje komponent.

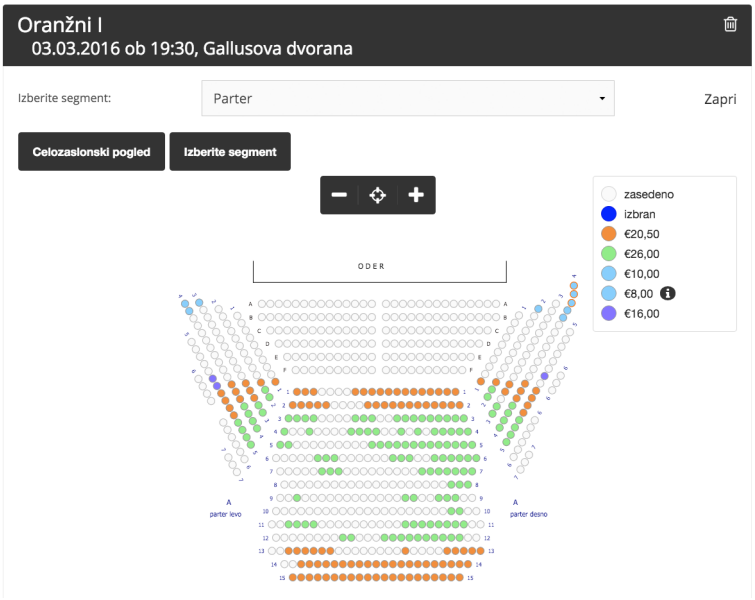
Postopno izboljševanje komponent smo v praksi uporabili pri načrtovanju komponente za izbiro termina in števila vstopnic na spletni strani mojekarte.si. Zgornja polovica slike 5.1 prikazuje osnovne HTML gradnike, ki omogočajo vse akcije, potrebne pri izbiri. Spodaj na sliki 5.1 pa je prikazana izboljšana različica komponente, ki se izriše v naprednejših spletnih brskalnikih.

Nekaterih komponent ni mogoče tako enostavno narediti dostopnih. Primer take komponente je komponenta za izbiro sedežev v dvorani. Grafična izbira sedeža je zaradi uporabljenih tehnologij povsem nedostopna enostavnejšim spletnim brskalnikom oz. bralnikom zaslona. Da bi lahko tudi uporabniki teh naprav in asistivnih tehnologij opravili nakup, je bilo potrebno razviti enostavnejšo alternativo. Enostavnejša alternativa predstavlja seznam gumbov s številom vstopnic. Uporabnik v tem primeru sedežev ne izbira s kliki na proste sedeže v izrisani dvorani, temveč izbere željeno število vstopnic s pomočjo gumbov. Ob izbiri števila vstopnic algoritem v ozadju poskrbi, da so v košarico dodane vstopnice za najboljše še proste sedeže v dvorani. Pri-

Slika 5.2: Osnovna različica komponente za izbiro sedežev.

mera obeh različic komponente za izbiro sedežev sta prikazana na slikah 5.2 in 5.3.

Dostopnost spletne strani uporabnikom asistivnih tehnologij se lahko še dodatno poveča z uporabo ustreznih HTML atributov. To so atributi *aria* (ang. Accessible Rich Internet Applications). [14] Njihova vloga je, da dodajo semantični pomen HTML elementom, katerih funkcionalnost je različna od osnovne funkcionalnosti, definirane s standardom HTML. Enostaven primer takega elementa je npr. sidro povezave *Gumb*, katerega osnovna funkcionalnost in izgled sta zamenjana s funkcionalnostjo in izgledom gumba (npr. HTML elementa *button*). Semantično pravilnost takega elementa se zagotovi z dodatnim *aria* atributom *role="button"*. Po-



Slika 5.3: Napredna različica komponente za izbiro sedežev.

leg številnih *aria* atributov za določanje semantičnega pomena elementov obstajajo tudi *aria* atributi za opis stanja komponente. To so atributi *aria-hidden*, *aria-expanded* in *aria-collapsed*. Navedeni atributi lahko brskalniku sporočajo ali je komponenta za uporabnika trenutno skrita, ali je komponenta prikazana v celoti oz. je del komponente skrit ipd. Pravilna uporaba *aria* atributov omogoči bralnikom zaslona, da naprava uporabniku te asistivne tehnologije prebere samo podatke, ki so bistveni za uporabo spletne strani.

Poglavje 6

Trajnost

Raznolikost spletnih brskalnikov, ki jih uporabniki uporabljajo za dostop do spletnih strani, je velika. Zato je nujno potreben sistem, s pomočjo katerega lahko zaznavamo, katere funkcionalnosti določen brskalnik podpira. Le tak sistem nam namreč omogoča, da spletno stran pravilno prilagodimo in tako uporabnikom različnih brskalnikov zagotovimo dobro uporabniško izkušnjo.

6.1 Zaznavanje naprav in brskalnikov

Prva zaznavanja podprtih funkcionalnosti so temeljila na razpoznavanju naprav in brskalnikov na podlagi t.i. *user agent* niza (v nadaljevanju UA niz). To je niz besedila, ki ga brskalnik pošlje strežniku in se z njim predstavi. Za brskalnik Google Chrome verzije 41 izgleda takole:

```
Mozilla/5.0 (Windows NT 6.4; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/41.0.2225.0 Safari/537.36
```

Iz zgornjega primera je razvidna glavna pomanjkljivost prilagajanja funkcionalnosti spletne strani na podlagi zaznavanja naprav in brskalnikov. Kljub temu da je niz del zahtevka, ki ga brskalnik Google Chrome 41 pošlje strežniku, so v nizu zapisane besede "Mozilla", "AppleWebKit" in "Safari". To pomeni, da se Googlov brskalnik strežniku ne predstavi zgolj kot Google

Chrome 41, temveč tudi kot brskalnik Mozilla Firefox in brskalnik Safai. Razlog za to je želja razvijalcev brskalnikov, da njihovi uporabniki ne bi bili prikrajšani za vsebino, ki bi jo spletni strežnik na podlagi UA niza serviral samo določeni podmnožici naprav in brskalnikov. Tako prirejanje UA nizov je precej izničilo njihov primarni namen in se zato dandanes za zaznavanje funkcionalnosti brskalnikov praviloma le redko uporablja.

Poleg omenjene pomanjkljivosti je tako zaznavanje naprav in brskalnikov odsvetovano tudi zato, ker otežuje razvoj in vzdrževanje spletne strani. Spletna stran za uporabo takega zaznavanja potrebuje zbirko podatkov o napravah in funkcionalnosti, ki jih te naprave podpirajo. Ta zbirka podatkov zahteva redno osveževanje, v nasprotnem primeru zaznavanje ne deluje pravilno. Zaznavanje naprav in brskalnikov se zato v praksi uporablja samo v primeru, ko odpovejo vse ostale tehnike zaznavanja funkcionalnosti (opisane v nadaljevanju poglavja 6), ki jih brskalnik podpira.

6.2 Preverjanje podpore funkcionalnosti

S testiranjem, ali brskalnik podpira neko funkcionalnost, odpravimo pomisleke, prisotne pri uporabi tehnike zaznavanja naprav in brskalnikov. Preverjanje se namreč izvaja v obratni smeri in od razvijalcev ne zahteva vzdrževanja zbirke podatkov o napravah in podprtih funkcionalnostih. Osnovna ideja tehnike preverjanja podpore funkcionalnosti je, da pred uporabo določene funkcionalnosti preverimo, ali brskalnik to funkcionalnost omogoča. V primeru brskalnika, ki funkcionalnost podpira, jo lahko uporabimo za obogatitev uporabniške izkušnje. V nasprotnem primeru stran za uporabnika še vedno ohranja osnovno funkcionalnost (upoštevanje vidikov uporabnosti ter dostopnosti, opisanih v poglavjih 4 in 5), le funkcionalnosti, ki bi izboljšale oz. popestrile uporabniško izkušnjo, niso na voljo.

V poglavju 3.3 smo že opisali medijske poizvedbe. Le-te predstavljajo način zaznavanja podprtih funkcionalnosti in lastnosti brskalnika v CSS kodi. V nadaljevanju bomo opisali še zaznavanje podprtih funkcionalnosti z upo-

rabo JavaScript kode, predstavili JavaScript knjižnico Modernizr, ki je na spletu široko uporabljana za zaznavanje podprtih funkcionalnosti brskalnikov, in opisali način, s katerim v brskalnikih, ki neke funkcionalnosti ne podpirajo, to funkionalnost podpreti z uporabo t.i. polifilov.

6.2.1 Zaznavanje podprtih funkcij z JavaScript kodo

Najpogosteje se testiranje podpore neke funkcionalnosti izvaja v JavaScript kodi. Razvijalci različnih spletnih brskalnikov različno hitro in v različnem vrstnem redu v brskalnike vključujejo nove funkcionalnosti. Neuporaba ustreznega testiranja bi povzročila nedelovanje spletne strani na množici brskalnikov, ki te funkcionalnosti ne podpirajo oz. bi razvoj spletne strani omejila samo na funkcionalnosti, podprte v vseh brskalnikih. Taka omejitev pa je zelo stroga, saj del uporabnikov spletne strani uporablja zastarele brskalnike, v katerih nove funkcionalnosti ne bodo nikoli na voljo. To posledično pomeni, da spletna stran tudi uporabnikom novejših in naprednejših brskalnikov ponuja neizboljšano uporabniško izkušnjo.

Z JavaScript kodo lahko preverjamo različne sklope funkcionalnosti. Najenostavneje je testirati, ali je v brskalnikovem JavaScript pogonu določena funkcija podprta. Če klic funkcije vrne vrednost *undefined*, to pomeni, da funkcija ni podprta in koda znotraj if-bloka se ne izvede. Primer take detekcije funkcionalnosti je prikazan v spodnji JavaScript kodi:

```
// Ce obstaja podpora za standardno metodo za dodajanje
// akcij k dogodkom, jo uporabi.
if (document.addEventListener) {
    document.addEventListener('click', callback);
}
// V nasprotnem primeru preizkusi nestandardno metodo
// "attachEvent" uporabljeno v starejsih verzijah Internet
// Explorer-ja.
else if (document.attachEvent) {
```

```
    document.attachEvent('onclick', callback);
}
```

Poleg funkcionalnosti JavaScript pogona v brskalniku lahko z JavaScript kodo zaznavamo tudi podporo za pravila CSS. Z uporabo CSS JavaScript API-ja je mogoče podporo za neko pravilo CSS testirati na način, prikazan v spodni JavaScript kodi: [15]

```
if (CSS.supports('(transition: none)')) {
    /* CSS prehodi so podprti. */
}
```

V kolikor sam CSS JavaScript API ni podprt v brskalniku, bo zgornja koda ignorirana in ne bo povzročala težav.

Testiranje ostalih lastnosti brskalnika je zahtevnejše in v nekaterih primerih zahteva uporabo precej domiselnih rešitev. Razlog za to tiči v tem, da za določanje teh lastnosti ne obstaja standardiziran JavaScript API, ki bi omogočal enostavno preverjanje lastnosti v JavaScript kodi.

Prikazali bomo primer domiselnega testiranja, ali brskalnik podpira HTML element *canvas*, namenjen izrisu grafik, animacij ipd. Spodnja JavaScript koda programsko ustvari nov *canvas* element in nato preveri, če ta element vsebuje funkcijo *getContext()*. V brskalnikih, ki risanja na *canvas* element ne podpirajo, preverjana funkcija ne obstaja. Po tem lahko sklepamo, ali brskalnik *canvas* element podpira ali ne.

```
function isCanvasSupported() {
    var elem = document.createElement('canvas');
    return !!(elem.getContext && elem.getContext('2d'));
}
```

Mnogo testov za preverjanje podpore funkcionalnosti zahteva še veliko več domiselnosti in bolj zapleteno JavaScript kodo. Zato je priporočljiva uporaba JavaScript knjižnic, ki raznorazne zapletene teste skrijejo za enostaven API,

ki ga kot razvijalci lahko uporabljamo za določanje podpore funkcionalnosti. Najbolj poznana taka knjižnica je Modernizr, ki jo bomo opisali v nadaljevanju.

6.2.2 Knjižnica Modernizr

JavaScript knjižnica Modernizr je zbirka testov, ki povedo, katere JavaScript, CSS in HTML funkcionalnosti podpira uporabnikov spletni brskalnik. [16] Razvijalcu omogoča detekcijo funkcionalnosti v JavaScript in CSS kodi.

Uporabo Modernizr-ovih testov v JavaScript kodi bomo prikazali na primeru izrisa dvorane na spletni prodajalni vstopnic mojekarte.si. Spodnja koda najprej preveri, ali brskalnik podpira tehnologijo SVG. V primeru, da podpore za SVG tehnologijo v brskalniku ni, se preveri, ali je podprta tehnologija Flash. Če nobena od naštetih tehnologij ni podprta, se dvorana ne izriše, temveč se uporabniku prikažejo gumbi za izbiro števila vstopnic, ki po principu "najboljšega še prostega sedeža" dodajo vstopnice v košarico (brez grafične izbire sedežev).

```
if (Modernizr.svg) {
    // Brskalnik podpira tehnologijo SVG.
}
else {
    // Brskalnik ne podpira tehnologije SVG.

    Modernizr.on('flash', function(result) {
        if (result) {
            // Brskalnik podpira tehnologijo Flash.
        }
        else {
            // Brskalnik ne podpira tehnologij SVG ali Flash.
        }
    });
```

```
}
```

Drugi način uporabe knjižnice Modernizr je detekcija funkcionalnosti v CSS kodi. Knjižnica Modernizr po končanem nalaganju samodejno izvede teste za podporo številnih funkcionalnosti. V primeru, da brskalnik neko funkcionalnost podpira, se HTML elementu *html* pripne razred z njenim imenom. Primer elementa *html* pred in po testiranju prikazuje slika 6.1. V CSS kodi lahko dodatna pravila, ki jih brskalnik podpira, elementu dodamo tako, da selektorju elementa na začetek pripnemo še selektor razreda za željeno pravilo. Spodnji primer CSS kode prikazuje, kako barvo ozadja elementa zapisati v RGBA obliki z definirano prosojnostjo, če brskalnik tak zapis podpira. V nasprotnem primeru se uporabi zapis RGB, ki ne podpira prosojnosti.

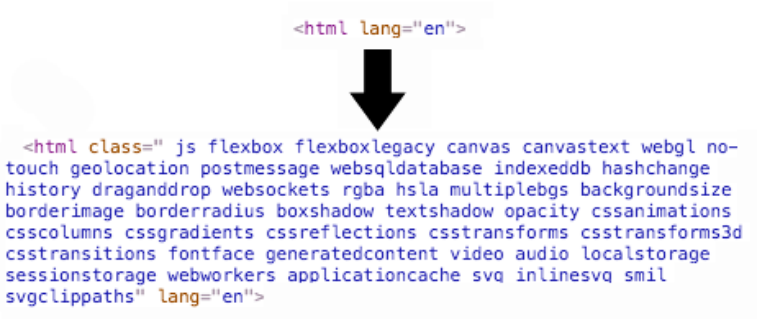
```
.element {
    background-color: rgb(255, 0, 0);
}

.rgba .element {
    background-color: rgba(255, 0, 0, 0.8);
}
```

6.2.3 Uporaba nepodprtih funkcionalnosti (polifili)

V nekaterih primerih je uporaba določene funkcionalnosti brskalnika potrebna za pravilno delovanje spletne strani. Delovanja namreč ne moremo enostavno prilagoditi tako, da bi bila spletna stran uporabna tudi za uporabnike brskalnikov, ki te funkcionalnosti ne podpirajo. V takih primerih lahko razvijalci uporabijo t.i. polifile.

Polifil je dodatna programska koda, ki jo razvijalec vključi v spletno stran. Deluje tako, da najpogosteje z domiselnimi rešitvami v brskalniku omogoči funkcionalnosti, ki jih ta v osnovi ne podpira. Polifile za večino funkcionalnosti, s katerimi je mogoče razširiti delovanje brskalnika, lahko najdemo na



Slika 6.1: Element *html* pred in po izvedbi testov vključenih v knjižnico Modernizr.

spletu, saj so zaradi vsesplošne uporabnosti to večinoma odprtokodne rešitve. Eden izmed seznamov polifilov je na voljo v GitHub repozitoriju knjižnice Modernizer. [17]

Primer zelo široko uporabljenega polifila je HTML5 Shiv. [18] Nekateri starejši brskalniki, ki so še vedno v uporabi, kot je npr. Internet Explorer 8, ne podpirajo HTML elementov, ki so bili dodani s standardom HTML5. To težavo razvijalec spletne strani zaobide tako, da v spletno stran vključi skripto HTML5 Shiv, ki brskalniku doda podporo za HTML elemente, navedene v novem standardu. Tako lahko razvijalec uporablja nove funkcionalnosti v brskalnikih in sledi trendom razvoja, pri tem pa ne zmanjšuje uporabnosti spletne strani za uporabnike starejših brskalnikov.

Poglavje 7

Zmogljivost

Zmogljivost spletne strani se pogosto enači s percepcijo hitrosti, s katero se vsebina in funkcionalnost spletne strani preneseta k uporabniku in odzivnostjo/hitrostjo delovanja uporabniškega vmesnika. S tem, ko spletno stran naredimo odzivno in prilagojeno za pravilno delovanje na širokem spektru naprav, optimizacije še zdaleč nismo zaključili. Spletne strani namreč nismo prilagodili za optimalen prenos podatkov s strežnika na uporabnikovo napravo. Pomanjkanje optimiziranega prenosa podatkov ne pride toliko do izraza na računalnikih in napravah s hitro povezavo na svetovni splet. Vzpostavljanje povezave in prenos podatkov preko mobilnega omrežja pa sta praviloma veliko počasnejša. Zato je lahko spletna stran (kljub prilagojenemu izgledu in funkcionalnosti) zaradi počasnega oz. neuspešnega nalaganja neuporabna za uporabnike, ki do spletne strani dostopajo preko mobilnih omrežij. V tem poglavju bomo opisali, kako spletno stran čimbolje prilagoditi omejitvam omrežja.

7.1 Omejitve zmogljivosti omrežja

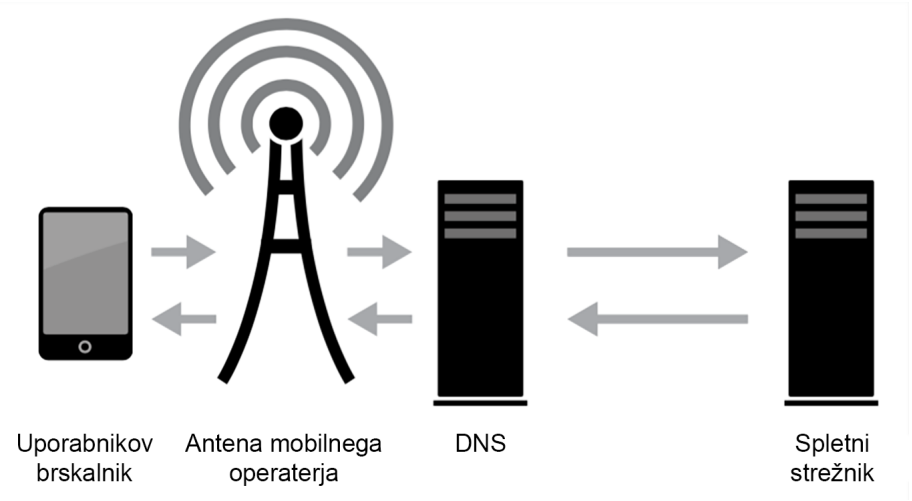
Da bi lahko težavo, opisano v uvodu poglavja 7, bolje razumeli, bomo na kratko predstavili potek prenosa spletne strani s strežnika na uporabnikovo napravo. Za komunikacijo strežnik in brskalnik uporabljata protokol HTTP.

V trenutku, ko uporabnik v brskalniku sproži zahtevo po nalaganju strani, brskalnik pošlje poizvedbo DNS strežniku, ki vpisani URL naslov preslika v dejanski IP naslov strežnika, na katerem se nahaja spletna stran. Šele nato lahko brskalnik pravilno pošlje zahteve na spletni strežnik in začne prenos datotek, potrebnih za prikaz spletne strani.

V primeru, da uporabnik za dostop do spletne strani uporablja mobilno omrežje, se zgoraj naštetim stopnjam doda še ena stopnja. Naprava najprej vzpostavi komunikacijo z anteno mobilnega operaterja, le-ta pa posreduje zahtevo naprej na DNS strežnik in tako omogoči izvedbo ostalih stopenj, opisanih v prejšnjem odstavku (stopnje povezave so prikazane na sliki 7.1). Ta radijska povezava med napravo in anteno mobilnega operaterja je zelo počasna v primerjavi z WiFi oz. ethernet povezavo. Ob uporabi tehnologije 3G potrebuje zahtevek v povprečju 2 sekundi za izvedbo, kar povzroči, da je nalaganje iste spletne strani prek mobilnega omrežja v povprečju 2 sekundi počasnejše kot pri nalaganju preko WiFi oz. ethernet povezave. [19] Novejše tehnologije 4G in LTE izdatno skrajšajo čas, potreben za prvo poizvedbo, njihova uporaba pa še ni razširjena in zato je nalaganje spletnih strani preko mobilnih omrežij že v osnovi počasno. Z optimizacijo težave počasne prve zahteve ne moremo rešiti, saj je to stvar delovanja mobilnih omrežij, lahko pa z ostalimi pospešitvami poskrbimo, da skupen čas nalaganja spletne strani za uporabnika ni predolg.

7.2 Blokada izrisa spletne strani

Opisali smo, kako brskalnik spletnemu strežniku posreduje zahtevo za spletno stran. Strežnik na to zahtevo odgovori s HTML dokumentom, ki spletno stran predstavlja. Brskalnik HTML dokument že med prenašanjem s strežnika začne razčlenjevati in v njem iskati povezave na dodatne datoteke, navedene v HTML dokumentu, ki so potrebne za izris spletne strani. To so npr. stilne predloge, datoteke z JavaScript kodo, slike, pisave ipd. Ko brskalnik naleti na referenco na dodatno datoteko, pošlje na strežnik zahtevo



Slika 7.1: Grafični prikaz povezave na splet preko mobilnega omrežja.

za prenos te datoteke. Ta prenos se izvaja asinhrono, medtem ko brskalnik nadaljuje z razčlenjevanjem osnovnega HTML dokumenta. Brskalnik po razčlenitvi osnovnega HTML dokumenta na podlagi strukture tega dokumenta izdela model za predstavitev objektov dokumenta (ang. document object model, v nadaljevanju DOM). Ko je model zgrajen, ga lahko funkcije v JavaScript kodi uporabljajo za iskanje in programsko spreminjanje HTML elementov na strani. Poleg uporabe v JavaScript kodi DOM služi tudi določanju izrisa elementov na podlagi stilnih predlog spletne strani.

Brskalnik z izrisom spletne strani začne, ko so zaključeni prenosi vseh stilnih predlog in JavaScript datotek, navedenih v osnovnem HTML dokumentu. Pravimo, da prenos CSS in JavaScript datotek blokira izris spletne strani. Glavni razlog za tako odločitev razvijalcev spletnih brskalnikov je v tem, da vsebina CSS datotek in JavaScript datotek tako spremeni izgled oz. vsebino spletne strani, da je v večini primerov potreben ponovni izris spletne strani takoj po končanem prenosu teh dodatnih datotek. V izogib večkratnim izračunom in risanju je bilo uvedeno pravilo o blokiranju izrisa spletne strani do konca prenosa in razčlenitve stilnih predlog in JavaScript datotek. [20]

Iz napisanega lahko razberemo, da brskalnik najhitreje izriše strani, ki ne vsebujejo referenc na dodatne CSS oz. JavaScript datoteke. Obstoj take strani je možen na dva načina:

- Spletna stran za svoje delovanje in izris ne potrebuje JavaScript kode oz. stilnih predlog. Izgled in funkcionalnost takih strani sta zelo omejena, zato so tako izdelane spletne strani zelo redke.

- Vsa JavaScript koda in CSS pravila so zapisana v glavi osnovnega HTML dokumenta z uporabo značk <*script*> in <*style*>. Glavna pomanjkljivost takega pristopa k razvoju spletnih strani je nezmožnost brskalnika, da bi prenešeno JavaScript in CSS kodo shranil v predpomnilnik. To posledično pomeni ponovni prenos iste kode ob vsakem ponovnem nalaganju strani, kar izdatno poveča količino prenesenih podatkov med napravo in spletnim strežnikom. Predvsem pri povezavah na splet preko mobilnih omrežij se ta pomanjkljivost odraža v počasnem nalaganju spletne strani in visoki porabi zakupljenega podatkovnega prenosa podatkov.

S kombinacijo tehnik, opisanih v nadaljevanju poglavja 7, je možno brskalnik pretentati, da spletno stran izriše, preden se zaključijo prenosi vseh dodatnih CSS in JavaScript datotek. Tako lahko uporabniku hitreje serviramo spletno stran, ki omogoča osnovni pregled vsebine in uporabo osnovnih funkcionalnosti.

7.3 Priprava datotek za prenos prek omrežja

Učinkovit in hiter prenos datotek, potrebnih za izris in delovanje spletne strani, dosežemo tako, da število prenesenih datotek zmanjšamo, kolikor je to mogoče, in da zmanjšamo velikost posameznih datotek, ki se prenašajo. V nadaljevanju si bomo pogledali, kako to dosežemo pri slikah ter besedilnih datotekah.

7.3.1 Orodje Grunt

Grunt [21] je orodje, ki omogoča avtomatsko izvajanje ponavljajočih se opravil pri razvoju spletnih strani. Deluje tako, da razvijalec na svoj sistem namesti orodje Grunt in vtičnike, namenjene avtomatizaciji posameznih opravil. Nato v datoteki *Gruntfile.js* definira, v katerih pogojih in v kakšnem zaporedju se določena opravila avtomatsko izvedejo.

7.3.2 Optimizacija slikovnih datotek

Prenos slik najbolje optimiziramo tako, da za prenos izberemo slike, ki so po velikosti najbolj primerne napravi, ki spletno stran prikazuje. Tehnika je podrobno opisana v poglavju 7.6.3.

Z uporabo nekaterih drugih tehnik pa lahko dosežemo še dodatno zmanjšanje količine prenesenih podatkov ob nalaganju slik. Poskrbeti moramo, da s pomočjo ustrezne kompresije dosežemo čimmanjše velikosti slikovnih datotek. Že med načrtovanjem izgleda spletne strani je dobro imeti v mislih, da je enostavnejše slike in grafike možno bolj stisniti kot tiste z veliko barvami in gradienti.

Stiskanje slikovnih datotek je razmeroma enostaven proces. Obstaja veliko programov, ki kompresijo slikovnih datotek avtomatsko izvedejo ob uvozu željenih datotek. V procesu razvoja spletne prodajalne vstopnic mojekarte.si smo v ta namen uporabili program Shrink O'Matic. [22] Obstaja še veliko drugih rešitev za stiskanje slikovnih datotek. Kot primer navedimo še program ImageOptim. [23]

Proces optimizacije slikovnih datotek lahko še dodatno poenostavimo z uporabo vtičnikov za orodje Grunt. Primer takega vtičnika je vtičnik grunt-contrib-imagemin. [24]

7.3.3 Optimizacija besedilnih datotek

Zmanjševanje števila datotek, ki se prenašajo med spletnim strežnikom in napravo, ne pomeni vedno brisanja datotek. Združevanje vsebine več CSS

Slika 7.2: Zaslonska slika programa Shrink O'Matic.

oz. JavaScript datotek v eno datoteko sicer pomeni, da bo velikost te datoteke večja, vendar prenos ene večje datoteke s strežnika poteka hitreje kot prenos več manjših datotek. Razlog je v tem, da je pri prenosu ene večje datoteke potrebno vzpostaviti povezavo samo enkrat, pri prenosu več datotek pa se povezava s strežnikom vzpostavi večkrat. Vsaka vzpostavitev povezave traja nekaj časa, zato je priporočljivo število zahtevkov, poslanih na strežnik, čimbolj zmanjšati.

Ker so datoteke s programsko kodo na produkcijskih spletnih straneh namenjene temu, da jih prebere in razčleni spletni brskalnik, jih lahko še dodatno zmanjšamo s tem, da iz njih izključimo vse nepotrebne komentarje, presledke in znake za nove vrstice. JavaScript kodo skrčimo še s spremembo imen spremenljivk in funkcij, tako da so le-ta čim krajša.

Omenjeni pristopi k zmanjševanju velikosti besedilnih datotek izrazito zmanjšajo berljivost kode v teh datotekah. Zato se jih v procesu razvoja ne poslužujemo. Pred objavo spletne strani v produkciji pa jih najpogosteje s pomočjo orodij (npr. Grunt) avtomatsko izvedemo.

Pri izdelavi prenovljenega čelnega sistema spletne prodajalne vstopnic

smo za združevanje ter stiskanje tekstovnih datotek uporabili sledeče vtičnike za orodje Grunt:

- grunt-contrib-concat
- grunt-contrib-uglify
- grunt-contrib-copy
- grunt-contrib-watch
- grunt-contrib-sass

Definirali smo opravilo, ki ob spremembi katerekoli JavaScript datoteke v direktoriju *js* in njegovih poddirektorijih izvede združevanje in stiskanje datotek. Rezultat je ena stisnjena JavaScript datoteka z vso programsko kodo. Podobno smo definirali tudi opravilo za združevanje in stiskanje stilnih predlog.

7.4 Gzip

Gzip je široko uporabljen način stiskanja vsebine, ki se s spletnega strežnika prenaša k uporabniku. Uporabnikov spletni brskalnik v zahtevi za spletno stran strežniku sporoči, katere metode stiskanja podpira. Najpogosteje je to metoda Gzip. V primeru, da strežnik podpira katero izmed naštetih metod stiskanja, se datoteke pred prenosom k uporabniku stisnejo z uporabo ene izmed podprtih metod. Tako stisnjene datoteke so praviloma veliko manjše. To pomeni, da se po omrežju prenaša manj podatkov, kar ima pozitivne posledice na hitrost nalaganja spletne strani.

Za stiskanje vsebine metoda Gzip uporablja algoritem Deflate. Koliko bo datoteke mogoče stisniti, je odvisno od vsebine datotek. Najbolje algoritem deluje za tekstovne datoteke, v katerih se določeni znaki oz. zaporedja velikokrat ponovijo. Velikost takih datotek je možno stisniti tudi za 70 %. Za že optimizirane slikovne datoteke v formatih JPG, PNG, GIF ipd. se uporaba

stiskanja odsvetuje, saj je le-to počasno in prihranki velikosti datotek niso veliki. Za grafike v SVG formatu pa je uporaba zelo priporočljiva, saj so datoteke v SVG formatu navadne (neoptimizirane) besedilne datoteke.

Konfiguracija Apache strežnika, ki za stiskanje tekstovnih datotek uporablja algoritem deflate, izgleda takole:

```
AddType image/svg+xml .svg

AddOutputFilterByType DEFLATE text/plain
AddOutputFilterByType DEFLATE text/html
AddOutputFilterByType DEFLATE text/xml
AddOutputFilterByType DEFLATE text/css
AddOutputFilterByType DEFLATE application/xml
AddOutputFilterByType DEFLATE application/xhtml+xml
AddOutputFilterByType DEFLATE application/rss+xml
AddOutputFilterByType DEFLATE application/javascript
AddOutputFilterByType DEFLATE application/x-javascript
AddOutputFilterByType DEFLATE image/svg+xml
```

7.5 Predpomnilnik brskalnika

Kljub temu da z ustrezno pripravo datotek dosežemo kar se da optimizirane velikosti le-teh za prenos preko omrežja, je bolje, da datotek prek omrežja sploh ne prenašamo, če to ni potrebno. S tem namenom je v brskalnik vgrajen predpomnilnik. Vanj se shranjujejo statične datoteke, prenesene s strežnika. To so datoteke, katerih vsebina se ne spreminja z vsakim ponovnim nalaganjem spletne strani. Primeri takih datotek so datoteke z JavaScript kodo, stilne predloge, slikovne datoteke ipd.

Spletni strežnik ob prvem obisku spletne strani brskalniku pove, katere datoteke lahko shrani v predpomnilnik in za koliko časa. Do poteka tega časa brskalnik ob ponovnem nalaganju strani na strežnik ne pošilja več zahtev za prenos predpomnjenih datotek, temveč jih naloži iz lokalnega predpomnil-

nika. To zelo skrajša čas, potreben za nalaganje spletne strani in zmanjša količino podatkov, prenesenih s spletnega strežnika.

Konfiguracija Apache strežnika, da za določene datoteke v glavo odgovora doda informacije, potrebne za predpomnjenje teh datotek v spletnem brskalniku, izgleda takole:

```
ExpiresActive On
ExpiresByType text/javascript "access plus 45 days"
ExpiresByType text/css "access plus 45 days"
ExpiresByType image/gif "access plus 45 days"
ExpiresByType image/jpeg "access plus 45 days"
ExpiresByType image/png "access plus 45 days"
```

Zgornja konfiguracija za JavaScript datoteke, stilne predloge in slike v različnih formatih nastavi čas predpomnjenja v spletnem brskalniku na 45 dni. Po preteku tega časa brskalnik datoteke ponovno prenese s spletnega strežnika in novo različico shrani v predpomnilnik.

7.6 Prenos datotek s strežnika k uporabniku

7.6.1 Prenos CSS

Stilne predloge predstavljajo velik del podatkov, ki se s spletnega strežnika prenašajo k uporabniku. Za izdelavo uporabnega uporabniškega vmesnika, ki tudi lepo izgleda, je velikokrat potrebna večja količina CSS kode. Poleg tega se količina potrebne CSS kode še povečuje z uporabo CSS pravil znotraj medijskih poizvedb, s katerimi spletne strani naredimo odzivne.

Boljšo preglednost velike količine CSS kode razvijalci pogosto dosežejo z modularnim pristopom k pisanju stilnih predlog. Stilno predlogo spletne strani razdelijo v več manjših in zato bolj preglednih datotek. Vsaka datoteka vsebuje CSS pravila za posamezni modul, funkcionalnost oz. komponento spletne strani.

V poglavju 7.3 smo zapisali, da je za optimalni prenos datotek prek omrežja potrebno poskrbeti, da je število datotek čim manjše in da so čim manjše tudi velikosti teh datotek. Serviranje več manjših datotek stilnih predlog ne upošteva tega priporočila v celoti, saj brskalnik za vsako posamezno stilno predlogo na strežnik pošlje novo HTTP zahtevo. Zato se je uveljavila tehnika združevanja stilnih predlog v eno datoteko s pomočjo avtomatskih procesov pred objavo spletne strani v produkciji. Datoteka z združenimi stilnimi predlogami je nato pogosto stisnjena še s strani spletnega strežnika pred pošiljanjem uporabniku, najpogosteje z uporabo metode Gzip. Z naštetimi koraki se upoštevajo priporočila o pripravi datotek za pošiljanje prek omrežja, saj stilne predloge predstavljajo samo eno, zelo močno stisnjeno datoteko.

Tudi prenos ene same stilne datoteke lahko na počasnih omrežnih povezavah zahteva veliko dragocenega časa. V tem času brskalnik zaradi blokade izrisa do konca prenosa CSS in JavaScript datotek, navedenih v glavi HTML dokumenta, uporabniku ne prikaže ničesar, čeprav je vsebina velikokrat že na voljo. Zato se vse bolj uveljavlja tehnika nalaganja stilnih predlog z JavaScript kodo, ki omenjeno težavo odpravlja. Tehnika temelji na kombinaciji dveh načinov vključevanja stilnih predlog v HTML dokument:

- značka <*style*> v glavi HTML dokumenta in
- referenca na stilno predlogo z značko <*link*> v glavi HTML dokumenta.

V glavo posameznega HTML dokumenta med znački <*style*> in </*style*> vstavimo najpomembnejša CSS pravila za izris spletne strani. Ta CSS pravila bodo s spletnega strežnika prenesena hkrati s HTML dokumentom, zato njihov prenos ne bo blokiral izrisa spletne strani. Če predpostavimo, da spletna stran optimizira tudi prenos JavaScript kode, kot bomo to opisali v poglavju 7.6.2, se lahko osnovna verzija spletne strani izriše takoj po prenosu osnovnega HTML dokumenta, saj brskalnik ne čaka več na prenos stilnih predlog in JavaScript kode v zunanjih datotekah. Uporabniku na ta način brskalnik zelo hitro izpiše iskano vsebino. Čeprav vsebina še ne izgleda najbolje, jo lahko začne uporabnik hitro prebirati, s tem pa se poveča uporabnikova

percepcija hitrosti nalaganja spletne strani.

Stilna predloga, ki definira končni izgled celotne spletne strani, se v času, ko uporabnik že prebira vsebino, v ozadju prenaša s strežnika. Po prenosu se s pomočjo JavaScrip kode v glavo osnovnega HTML dokumenta vrine referenca na preneseno stilno predlogo. Spletni brskalnik nato izgled spletne strani prilagodi CSS pravilom iz vrinjene stilne predloge in nalaganje spletne strani se zaključi. Omenjena tehnika v resnici ne skrajša časa, potrebnega za nalaganje spletne strani, se pa zaradi mogoče zgodnje interakcije s spletno stranjo skrajša uporabnikovo čakanje na vsebino, zato se spletna stran navidezno hitreje naloži.

Uporabo zgoraj opisane tehnike v praksi prikazuje spodnja HTML koda v glavi HTML dokumenta:

```
<style>
    /* Najpomembnejsa CSS pravila za osnoven prikaz vsebine. */
</style>
<script>
    // Definicija funkcije za asinhrono nalaganje stilnih
    // predlog.
    function loadCSS(href){ ... }

    // Klic zgornje funkcije z referenco na stilno predlogo,
    // ki jo zelimo naloziti asinhrono.
    loadCSS('stil.css');
</script>
```

Programska koda funkcije za asinhrono nalaganje stilnih predlog loadCSS je prosto dostopna na GitHub repozitoriju, njen avtor je Scott Jehl. [25]

Ker je ročno vstavljanje in vzdrževanje osnovnih CSS pravil v glavi več HTML dokumentov zamudno, je priporočena uporaba orodij, kot je npr. vtičnik za Grunt grunt-criticalcss. [26] Vtičnik v procesu priprave spletne strani za objavo na produkcijskem strežniku avtomatsko v glave HTML dokumentov vstavi CSS pravila, ki so potrebna za osnovni izris posameznega

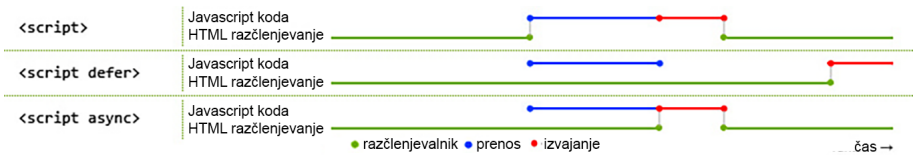
HTML dokumenta. To razvijalcem omogoči uporabo prej opisane tehnike z zelo malo dodatnega dela.

7.6.2 Prenos JavaScript kode

V poglavju 7.2 smo opisali, kako prenos zunanjih stilnih predlog in JavaScript kode blokira izris spletne strani. Brskalnik predpostavlja, da bo JavaScript koda spremenila vsebino oz. izgled strani, zato spletno stran izriše, ko se prenos kode zaključi in se koda izvede. Taka predpostavka brskalnika je povsem na mestu, ne drži pa vedno. Ob uporabi tehnike združevanja vseh JavaScript datotek v eno datoteko namreč veliko kode iz zdužene datoteke ne vpliva na vsebino in izgled naloženega HTML dokumenta. Zato bi se lahko taka koda naložila v ozadju, medtem ko uporabnik že prebira vsebino. Tako kot pri zakasnjenem nalaganju nekritičnih stilnih predlog bi tudi zakasnjeno nalaganje JavaScripta, ki ne blokira izrisa spletne strani, skrajšalo zaznani čas nalaganja spletne strani.

V ta namen v standardu HTML obstaja atribut *defer*, s standardom HTML5 pa je bil še dodatno uveden atribut *async*. Atributa dajeta brskalniku vedeti, da JavaScript koda v zunanji datoteki ne spreminja vsebine oz. izgleda strani. Brskalnik lahko zato izriše spletno stran brez čakanja na zaključek prenosa JavaScript kode.

Razlike v nalaganju strani ob uporabi različnih atributov so prikazane na shemi 7.3. [27] Tako atribut *defer* kot tudi atribut *async* povzročita asinhrono nalaganje JavaScript kode. Razlikujeta se v tem, kdaj se prenesena koda izvede. Skripte, naložene z uporabo atributa *defer*, se izvedejo po končanem razčlenjevanju HTML dokumenta (po zaporedju, ki ga definira zaporedje, po katerem so zunanje JavaScript datoteke vključene v HTML dokument). Skripte z uporabo atributa *async* pa se izvedejo takoj po končanem prenosu (zaporedje izvajanja skript je odvisno od tega, kdaj se zaključi njihov prenos, in je zato pogosto nepredvidljivo).



Slika 7.3: Pregled poteka nalaganja in izvajanja JavaScript kode.

7.6.3 Prenos slik

V povprečju 64 % teže spletnih strani v bajtih predstavljajo sike. [28] Na večjih zaslonih je tako razmerje med slikami in ostalimi datotekami razumljivo, saj večje slike potrebujejo več bajtov za svoj zapis. Na napravah z manjšimi zasloni pa ta podatek opozarja na veliko brez potrebe prenesenih bajtov.

Spletne strani, ki vsem napravam, ne glede na velikost zaslona, prikazujejo enako velike slike, niso zares odzivne. Vse ostale tehnike za izdelavo optimiziranih in odzivnih spletnih strani, opisane v diplomski nalogi, tako v veliki meri izgubijo svoj pomen. Uporabnik mora namreč ob nalaganju spletne strani prenesti velike količine podatkov (slik), kar pomeni počasno nalaganje strani in veliko porabo zakupljenega prenosa podatkov pri mobilnem operaterju kljub upoštevanju vseh ostalih prilagoditev, opisanih v diplomski nalogi.

Vstavljanje različno velikih slik v HTML dokument

Kot odgovor na opisano težavo je bil v HTML standard uveden način za prenašanje in prikazovanje različno velikih slik na napravah z različno velikimi zasloni. Način temelji na uporabi HTML elementa *img* z novima atributoma *srcset* in *sizes*. Kako v HTML dokument vključiti take odzivne slike, prikazuje spodnji izsek kode:

```
<img
    srcset="slike/velika-slika.jpg 1280w,
            slike/srednja-slika.jpg 640w,
```

```
        slike/mala-slika.jpg 320w"
sizes="(min-width: 40em) 40em,
       100vw"
src="slike/srednja-slika.jpg"
alt="Primer odzivne slike." />
```

Atribut *srcset* brskalniku poda seznam vseh različic slike z njihovimi absolutnimi širinami. Izvorna širina slike *velika-slika.jpg* je 1280 px, slike *srednja-slika.jpg* je 640 px in slike *mala-slika.jpg* 320 px.

Atribut *sizes* brskalniku poda informacije o tem, kakšna bo širina slike ob izrisu. V zgornjem primeru smo brskalniku podali dva podatka:

- na zaslonih širine vsaj 40 em, bo širina slike omejena na največ 40 em,
- na ožjih zaslonih bo širina slike enaka stoodstotni širini brskalniškega okna (ang. 100 % viewport width)

Te podatke brskalnik uporabi za izračun koeficienta primernosti posamezne slike iz seznama *srcset* in pošlje na strežnik zahtevo za najprimernejšo sliko. Pri izračunu brskalnik upošteva številne lastnosti naprave (velikost zaslona, resolucijo ipd.) in morebitne uporabnikove nastavitve o kvaliteti prenesenih slik, povečavi spletnih strani ipd.

Uporaba opisane tehnike omogoči serviranje najbolj primernih slik glede na lastnosti in nastavitve naprave. To v večini primerov pomeni velike prihranke pri količini prenesenih podatkov med spletnim strežnikom in uporabnikovo napravo in spletno stran naredi resnično odzivno. [30]

Za starejše brskalnike, ki novih HTML atributov *srcset* in *sizes* ne podpirajo, obstaja polifil Picturefill [29], ki tudi v teh brskalnikih omogoči uporabo zgoraj opisane tehnike. Če tudi delovanje polifila odpove, pa se v spletno stran vstavi slika definirana z atributom *src*. V zgornjem primeru je to slika *srednja-slika.jpg*.

Uporaba različno velikih slik v stilnih predlogah

V spletno stran lahko slike vstavimo tudi s CSS pravili, zato je pomembno poskrbeti, da so tudi tako vstavljene slike najprimernejših velikosti. V ta namen se uporabljajo medijske poizvedbe. Znotraj različnih blokov medijskih poizvedb lahko za isti element nastavimo povezave do slik različnih velikosti. Tako lahko glede na medijsko poizvedbo v spletno stran vstavimo najprimernejšo sliko.

V spodnjem primeru smo elementu *.element* nastavili za ozadje sliko s širino 320 px. Na napravah z zasloni, širšimi od 320 px, bo nastavitev ozadja prepisana s povezavo do slike z dvakratno širino, torej 640 px. S tem smo dosegli, da naprave s strežnika prenesejo samo najprimernejšo sliko. Naprave z manjšim zaslonom torej prenesejo manjšo sliko, kar posledično pomeni prihranek pri količini prenesenih podatkov.

```
.element {
    width: 100%;
    max-width: 320px;
    height: 300px;
    background-image: url(slike/mala-slika.jpg);
    background-size: 100% auto;
    background-repeat: no-repeat;
}

@media all and (min-width: 321px) {
    .element {
        max-width: 640px;
        background-image: url(slike/srednja-slika.jpg);
    }
}
```

Poglavje 8

Sklepne ugotovitve

V diplomskem delu smo opisali strategije in navedli najboljše prakse, s katerimi je mogoč razvoj spletnih strani, ki bodo pravilno delovale na čim širšem spektru naprav. Razvoj sodobnih spletnih strani zahteva posebno pozornost pri izzivih, ki se pojavijo zaradi spremenljivih pogojev omrežnih povezav in velikega števila različnih naprav, na katerih morajo spletne strani delovati brez večjih napak. Poleg izdelave prilagodljivega izgleda spletne strani je potrebno poskrbeti tudi za njeno uporabnost, s tem da uporabniški vmesnik prilagodimo lastnostim različnih naprav in interakcij z napravami. Med razvojem je potrebno poskrbeti tudi za dostopnost spletne strani, tako da lahko uporabniki vseh naprav, brskalnikov in asistivnih tehnologij dostopajo in razumejo funkcije ter vsebino spletne strani. V času, ko se tehnologije svetovnega spleta tako hitro razvijajo, je pomembno zagotoviti tudi trajnost spletne strani. Trajnost pomeni zagotovilo, da tehnologije, uporabljene za poganjanje spletne strani, pravilno delujejo danes in da bodo zagotavljale uporabne in dostopne spletne strani tudi uporabnikom v prihodnosti. Za dobro uporabniško izkušnjo je pomembna tudi zmogljivost spletne strani. Zmogljive spletne strani so strani, ki se hitro naložijo in katerih uporabniški vmesniki dajejo občutek odzivnosti in hitrega delovanja. V diplomskem delu smo predstavili, kako pri razvoju upoštevati in čim bolje zadostiti vsem tem vidikom odzivnih in optimiziranih spletnih strani. Navedene tehnike smo

upoštevali tudi pri razvoju čelnega dela sistema spletne platforme za prodajo vstopnic mojekarte.si in s tem zelo povečali uporabnost, odzivnost in dostopnost spletne prodajalne vstopnic.

Hiter razvoj spletnih tehnologij bo v prihodnosti povzročil, da bodo nekatere dobre prakse, opisane v diplomskem delu, postale nepotrebne, pojavili pa se bodo novi priporočeni načini izdelave spletnih strani. Primer takega razvoja, ki bo v naslednjih letih zagotovo spremenil način posredovanja podatkov s strežnika do uporabnika, je nova verzija protokola HTTP. Protokol HTTP/2 bo zaradi sprememb načina pošiljanja datotek prek omrežja odpravil potrebo po serviranju čim manjšega števila datotek pri nalaganju spletne strani.

Kljub tem spremembam pa bodo osnovni vidiki odzivnih in optimiziranih spletnih strani ostali enaki. Spletne strani bodo morale biti ne glede na tehnologije uporabne, dostopne, trajne in zmogljive. Razvoj spletnih tehnologij bo, tako upamo, spletnim razvijalcem samo olajšal izdelavo spletnih strani, ki bodo ustrezale tem pogojem.

Literatura

[1] S. Jehl, "Responsible Responsive Design", New York: A Book Apart, 2014

[2] E. Marcotte, "Responsive Web Design", New York: A Book Apart, 2011

[3] (2014) The birth of the web. Dosegljivo: http://home.web.cern.ch/topics/birth-web. [Dostopano 19. 12. 2015]

[4] (2015) Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update 2014–2019 White Paper. Dosegljivo: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_ec11-520862.html. [Dostopano 21. 12. 2015]

[5] (2013) Android Fragmentation Visualized. Dosegljivo: http://opensignal.com/reports/fragmentation-2013/. [Dostopano 21. 12. 2015]

[6] (2015) About JavaScript. Dosegljivo: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript/. [Dostopano 22. 12. 2015]

[7] (2007) Hypertext Transfer Protocol – HTTP/1.1. Dosegljivo: https://www.w3.org/Protocols/HTTP/1.1/rfc2616bis/draft-lafon-rfc2616bis-03.html. [Dostopano 23. 12. 2015]

[8] (2015) CSS Device Adaptation Module Level 1.
Dosegljivo: http://www.w3.org/TR/css-device-adapt/#the-viewport-rule. [Dostopano 20. 12. 2015]

[9] (2015) Responsive Web Design - The Viewport.
Dosegljivo: http://www.w3schools.com/css/css_rwd_viewport.asp. [Dostopano 20. 12. 2015]

[10] (2015) CSS3 @media Rule.
Dosegljivo: http://www.w3schools.com/cssref/css3_pr_mediaquery.asp. [Dostopano 26. 12. 2015]

[11] (2015) RWD.
Dosegljivo: http://interactivedesign.co.za/wp-content/uploads/2013/11/rwd1.png. [Dostopano 26. 12. 2015]

[12] (2015) A practical guide to web typography.
Dosegljivo: http://webtypography.net/. [Dostopano 28. 12. 2015]

[13] (2003) 3-D Finite-Element Models of Human and Monkey Fingertips to Investigate the Mechanics of Tactile Sense.
Dosegljivo: http://touchlab.mit.edu/publications/2003_009.pdf. [Dostopano 2. 1. 2016]

[14] (2016) WAI-ARIA Overview.
Dosegljivo: https://www.w3.org/WAI/intro/aria. [Dostopano 2. 1. 2016]

[15] (2016) CSS Conditional Rules Module Level 3.
Dosegljivo: https://drafts.csswg.org/css-conditional/#at-supports. [Dostopano 3. 1. 2016]

[16] (2016) Modernizr.
Dosegljivo: https://modernizr.com/. [Dostopano 3. 1. 2016]

[17] (2016) HTML5 Cross Browser Polyfills.
Dosegljivo: https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-Browser-Polyfills. [Dostopano 3. 1. 2016]

[18] (2016) The HTML5 Shiv.
Dosegljivo: https://github.com/afarkas/html5shiv. [Dostopano 3. 1. 2016]

[19] (2011) Making a mobile connection.
Dosegljivo: http://www.stevesouders.com/blog/2011/09/21/making-a-mobile-connection/. [Dostopano 5. 1. 2016]

[20] (2015) Critical rendering path.
Dosegljivo: https://developers.google.com/web/fundamentals/performance/critical-rendering-path/. [Dostopano 8. 1. 2016]

[21] (2016) Grunt.
Dosegljivo: http://gruntjs.com/. [Dostopano 8. 1. 2016]

[22] (2016) Shrink O'Matic.
Dosegljivo: http://toki-woki.net/p/Shrink-O-Matic/. [Dostopano 10. 1. 2016]

[23] (2016) ImageOptim.
Dosegljivo: https://imageoptim.com/. [Dostopano 10. 1. 2016]

[24] (2016) Grunt imagemin plugin.
Dosegljivo: https://github.com/gruntjs/grunt-contrib-imagemin. [Dostopano 10. 1. 2016]

[25] (2016) LoadCSS.
Dosegljivo: https://github.com/filamentgroup/loadCSS/. [Dostopano 12. 1. 2016]

[26] (2016) Grunt criticalcss plugin.
Dosegljivo: https://github.com/filamentgroup/grunt-criticalcss/. [Dostopano 12. 1. 2016]

[27] (2010) Asynchronous and deferred JavaScript execution explained.
Dosegljivo: http://peter.sh/experiments/asynchronous-and-deferred-javascript-execution-explained/. [Dostopano 14. 1. 2016]

[28] (2016) Interesting stats.
Dosegljivo: http://httparchive.org/interesting.php#bytesperpage. [Dostopano 14. 1. 2016]

[29] (2016) Picturefill.
Dosegljivo: http://scottjehl.github.io/picturefill/. [Dostopano 15. 1. 2016]

[30] (2014) Responsive Images in Practice.
Dosegljivo: http://alistapart.com/article/responsive-images-in-practice. [Dostopano 15. 1. 2016]